(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
15 November 2001 (15.11.2001)

PCT

(10) International Publication Number
WO 01/86430 A2

(51) International Patent Classification[7]: G06F 9/30

(21) International Application Number: PCT/US01/15180

(22) International Filing Date: 10 May 2001 (10.05.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/203,465 11 May 2000 (11.05.2000) US
60/203,409 11 May 2000 (11.05.2000) US

(71) Applicants *(for all designated States except US)*: **NETOCTAVE, INC.** [US/US]; 507 Airport Boulevard, Suite 111, Morrisville, NC 27560 (US). **HANNA, Michael** [US/US]; 522 N. Person Street, Raleigh, NC 27604 (US).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **BLAKER, David** [US/US]; 109 Hogan Glen Court, Chapel Hill, NC 27516 (US). **SAVARDA, Raymond** [US/US]; 4224 Sancroft Drive, Apex, NC 27502 (US).

(74) Agent: **MYERS BIGEL SIBLEY & SAJOVEC**; P.O. Box 37428, Raleigh, NC 27627 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

WO 01/86430 A2

(54) Title: CRYPTOGRAPHIC DATA PROCESSING SYSTEMS, COMPUTER PROGRAM PRODUCTS, AND METHODS OF OPERATING SAME IN WHICH A SYSTEM MEMORY IS USED TO TRANSFER INFORMATION BETWEEN A HOST PROCESSOR AND AN ADJUNCT PROCESSOR

(57) Abstract: Embodiments of cryptographic data processing systems, computer program products, and methods of operating same are provided in which system memory is used to transfer information between a host processor and an adjunct processor.

CRYPTOGRAPHIC DATA PROCESSING SYSTEMS, COMPUTER PROGRAM PRODUCTS, AND METHODS OF OPERATING SAME IN WHICH A SYSTEM MEMORY IS USED TO TRANSFER INFORMATION BETWEEN A HOST PROCESSOR AND AN ADJUNCT PROCESSOR

CROSS-REFERENCE TO PROVISIONAL APPLICATIONS

This application claims the benefit of Provisional Application Serial No. 60/203,409, filed May 11, 2000, entitled *Cryptographic Acceleration Methods and Apparatus*, and Provisional Application Serial No. 60/203,465, filed May 11, 2000, entitled *Methods and Apparatus for Supplying Random Numbers*, the disclosures of which are hereby incorporated herein by reference in their entirety as if set forth fully herein.

BACKGROUND OF THE INVENTION

The present invention relates generally to the field of data processing systems, and, more particularly, to cryptographic data processing systems, computer program products, and methods of operating same.

Signal processors and integrated circuit chips have been developed to accelerate cryptographic operations, such as public key operations. Examples of such chips include, but are not limited to, the Hifn 6500 available from Hifn, Inc., the SafeNet ADSP 2141 available from SafeNet, Inc., and the Rainbow Mykotronx FastMAP available from Rainbow Mykotronx, Inc. Despite the availability of cryptographic accelerator products, there remains room for improvement in the art.

For example, conventional cryptographic data processing systems generally use two main methods for issuing a command to a cryptographic accelerator: The first method involves the provision of a command register on the cryptographic accelerator that a host processor uses to issue a single command. Once the cryptographic

accelerator completes executing a command, the host processor may issue a new command. After completing a command, the cryptographic accelerator is generally idle until the host processor issues a new command. Unfortunately, the host processor may spend much time interacting directly with the cryptographic accelerator to download data and issue commands. This may reduce the amount of time available to the host processor for attending to other tasks.

The second method allows the host processor to download one or more command sequences to the cryptographic accelerator and then to instruct the cryptographic accelerator to execute one or more of the downloaded command sequences. After completing a command sequence, the cryptographic accelerator is generally idle until the host processor issues a new command. The size of the command sequences may be limited based on the amount of memory that may be placed on the cryptographic accelerator. Like the first method, the host processor may spend much time interacting directly with the cryptographic accelerator to download data and issue command sequences. This may reduce the amount of time available to the host processor for attending to other tasks.

Cryptographic accelerators generally perform operations using one or more operands. These devices may include general-purpose operand storage that comprises fixed length registers to store the operands and results. To execute an instruction, a register number is used to indicate which operand should be used for the operation and where the output should be stored. For example, if the operation were "a + b = c," then part of the instruction would indicate that "a" is in register 7, "b" is in register 1, and "c" should be put into register 2.

Because the registers are fixed in size and the operands and results are variable in size, the size of the operands will always be less than or equal to the register size. As a result, some of the memory in the registers may be wasted. This reduces the number of operands that may be stored on a chip in a given amount of space. In addition, if the cryptographic accelerator is redesigned to accommodate larger operands, then each of the registers may need to be modified. More registers may be designed into a cryptographic accelerator; however, adding more memory to a cryptographic accelerator may reduce the amount of other functionality that may be included and/or increase the cost.

Cryptographic processors and/or other types of signal processors and integrated circuits may use a hardware-based random number generator. Various

conventional methods may be used to retrieve random numbers from an integrated circuit incorporating a random number generator. One method is for the random number generator to provide one or more data registers that a host processor may read to obtain random numbers. The host processor may tell the random number generator to provide more random data before or after retrieving random data from the registers. The random number generator may generate the random data in the background so that random data may be available when needed by the host processor.

Another method for obtaining random data is for the host processor to request a sample of random data from the random number generator. The host processor may provide the random number generator with a request that specifies an amount of random data and a location in memory where the random data should be placed. The random number generator may then generate the random data and transfer the random data to the requested location in the background.

Unfortunately, by providing random data through data registers on the random number generator or other integrated circuit chip, any buffer management that may be desired is generally performed by the host processor. Moreover, the bus that connects the host processor with the random number generator may be used inefficiently because single data reads are typically used instead of block reads. If a host processor requests a block of random data, however, then the host processor may initiate the data transfers and any desired buffer management that may be desired is generally performed by the host processor. The foregoing operations may be performed in the background and/or a fast host processor may be used; however, a faster host processor may increase system costs.

SUMMARY OF THE INVENTION

Embodiments of the present invention provide cryptographic data processing systems, computer program products, and methods of operating same in which system memory is used to transfer information between a host processor and an adjunct processor. For example, in accordance with embodiments of the present invention, cryptographic data processing systems comprise a host processor, a system memory coupled to the host processor, and a cryptographic processor integrated circuit that comprises a local memory. One or more operands are downloaded into the local memory from the system memory, using, for example, a load command, and one or more operations are performed on at least one of the downloaded operands to generate

a result in the local memory. The result that is generated in the local memory is then stored in the system memory, using, for example, a store command. Advantageously, interaction between the host processor and the cryptographic processor may be reduced as the host processor need not consume processing time downloading operands to the cryptographic accelerator processor and/or uploading results from the cryptographic processor to system memory.

In accordance with further embodiments of the present invention, interaction between the host processor and the cryptographic processor may be further reduced and overall system performance improved by providing a command queue in the system memory, loading a command block into the command queue using the host processor, executing the command block using the cryptographic processor, and notifying the host processor that the command block has been executed. By loading commands into a command queue in the system memory where the cryptographic processor may retrieve them for processing, the host processor need not spend time interacting directly with the cryptographic processor.

In accordance with further embodiments of the present invention, the host processor may be notified that the command block has been executed in various ways. For example, the cryptographic processor may invoke an interrupt to notify the host processor that the command block has been executed. In particular embodiments, the interrupt may be invoked if the host processor has requested notification via an interrupt in an interrupt field of the command block. In other embodiments, the cryptographic processor may update a completion field in the command block to notify the host processor that the command block has been executed. In still other embodiments, a periodic interrupt may be defined such that when the interrupt occurs the host processor reads the completion fields of any command blocks.

In accordance with still further embodiments of the present invention, improved utilization of the system memory may be attained by re-using at least a portion of a command block that contains input data to store a result or output that is generated by an adjunct processor, such as the cryptographic processor. For example, a command queue may be provided in the system memory and a command block may be loaded into the command queue using the host processor. The command block comprises an input data field that contains input data. The adjunct processor performs an operation based on the input data to generate a result and this result is stored in the input data field such that at least a portion of the input data is overwritten.

Advantageously, the memory reserved for the command block in the system memory may be reduced because additional storage space need not be reserved to store the result of executing the command block either in the command block or elsewhere in the system memory.

Cryptographic processors and/or other types of signal processors and integrated circuits may use a hardware-based random number generator. In accordance with further embodiments of the present invention, random number samples may be provided for use by the host processor while reducing interaction between the host processor and the cryptographic processor. For example, a random number data queue in the system memory may be provided that has a read address and a write address associated therewith. The cryptographic processor loads a random number sample into the random number data queue at the write address and the host processor reads the random number sample beginning at the read address. The host processor need only interact with the cryptographic processor to update the read address and to check the value of the write address when the read address approaches the last value the host processor has for the write address. In addition, the cryptographic accelerator processor may manage the buffering of the random number samples, which may conserve processor cycles of the host processor.

BRIEF DESCRIPTION OF THE DRAWINGS

Other features of the present invention will be more readily understood from the following detailed description of specific embodiments thereof when read in conjunction with the accompanying drawings, in which:

FIG. 1 is a block diagram that illustrates cryptographic data processing systems, computer program products, and methods of operating same in accordance with embodiments of the present invention;

FIG. 2 is a flowchart that illustrates operations of cryptographic data processing systems and computer program products in accordance with embodiments of the present invention;

FIGS. 3 - 5 are block diagrams that illustrate functional execution units of a cryptographic accelerator processor in accordance with embodiments of the present invention;

FIG. 6 is a flowchart that illustrates operations of cryptographic data processing systems and computer program products in accordance with further embodiments of the present invention;

FIG. 7 - 8 are block diagrams that illustrate an encryption/authentication command queue and a public key command queue, respectively, in accordance with embodiments of the present invention;

FIGS. 9 - 11 are flowcharts that illustrate operations of cryptographic data processing systems and computer program products in accordance with further embodiments of the present invention;

FIGS. 12A - 12D are block diagrams that illustrate command blocks in accordance with embodiments of the present invention;

FIG. 13 is a flowchart that illustrates operations of cryptographic data processing systems and computer program products in accordance with further embodiments of the present invention;

FIGS. 14A, 14B, and **15** are block diagrams that illustrate command blocks in accordance with further embodiments of the present invention;

FIGS. 16 and **17** are flowcharts that illustrate operations of cryptographic data processing systems and computer program products in accordance with further embodiments of the present invention;

FIG. 18 is a block diagram that illustrates a random number generator data queue in accordance with embodiments of the present invention;

FIG. 19 is a flowchart that illustrates operations of cryptographic data processing systems and computer program products in accordance with further embodiments of the present invention;

FIG. 20 is a block diagram that illustrates a command interface for a conventional application specific integrated circuit;

FIG. 21 is a block diagram that illustrates parallel command interfaces for an application specific integrated circuit in accordance with embodiments of the present invention;

FIG. 22 is a block diagram of a cryptographic accelerator processor in which command interface managers are respectively associated with functional execution units in accordance with embodiments of the present invention; and

FIG. 23 is a flowchart that illustrates operations of cryptographic data processing systems and computer program products in accordance with further embodiments of the present invention.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

While the invention is susceptible to various modifications and alternative forms, specific embodiments thereof are shown by way of example in the drawings and will herein be described in detail. It should be understood, however, that there is no intent to limit the invention to the particular forms disclosed, but on the contrary, the invention is to cover all modifications, equivalents, and alternatives falling within the spirit and scope of the invention as defined by the claims. Like reference numbers signify like elements throughout the description of the figures. It will also be understood that when an element is referred to as being "connected" or "coupled" to another element, it can be directly connected or coupled to the other element or intervening elements may also be present. In contrast, when an element is referred to as being "directly connected" or "directly coupled" to another element, there are no intervening elements present.

The present invention may be embodied as methods, data processing systems, and/or computer program products. Accordingly, the present invention may be embodied in hardware and/or in software (including firmware, resident software, micro-code, *etc.*). Furthermore, the present invention may take the form of a computer program product on a computer-usable or computer-readable storage medium having computer-usable or computer-readable program code embodied in the medium for use by or in connection with an instruction execution system. In the context of this document, a computer-usable or computer-readable medium may be any medium that can contain, store, communicate, propagate, or transport the program for use by or in connection with the instruction execution system, apparatus, or device.

The computer-usable or computer-readable medium may be, for example but not limited to, an electronic, magnetic, optical, electromagnetic, infrared, or semiconductor system, apparatus, device, or propagation medium. More specific examples (a nonexhaustive list) of the computer-readable medium would include the following: an electrical connection having one or more wires, a portable computer diskette, a random access memory (RAM), a read-only memory (ROM), an erasable

programmable read-only memory (EPROM or Flash memory), an optical fiber, and a portable compact disc read-only memory (CD-ROM). Note that the computer-usable or computer-readable medium could even be paper or another suitable medium upon which the program is printed, as the program can be electronically captured, via, for instance, optical scanning of the paper or other medium, then compiled, interpreted, or otherwise processed in a suitable manner, if necessary, and then stored in a computer memory.

Referring now to **FIG. 1**, an exemplary cryptographic data processing system **12**, in accordance with embodiments of the present invention, comprises a cryptographic accelerator processor **14**, a host processor **16**, a cache memory **18**, a system memory **22**, and a system bus controller **24**, such as a north-bridge system controller. The system bus controller **24** couples the host processor **16** to the cache memory **18** and the system memory **22**, and also couples the host processor **16** and the system memory **22** to the cryptographic accelerator processor **14** via a system bus **26**, which may be, for example, a peripheral component interconnect (PCI) bus. The host processor **16** may be, for example, a commercially available or custom microprocessor. The system memory **22** is representative of an overall hierarchy of memory devices containing the software and data used to implement the functionality of the cryptographic data processing system **12**. The system memory **22** may include, but is not limited to, the following types of devices: ROM, PROM, EPROM, EEPROM, flash, SRAM, and DRAM.

In accordance with embodiments of the present invention, the cryptographic accelerator processor **14** comprises a random number generator (RNG) execution unit **28**, an encryption/authentication (E/A) execution unit **32**, and a public key (PK) engine execution unit **34**, which are coupled to a local memory **36** via a local bus **38**. In accordance with particular embodiments of the present invention, the system memory **22** contains a random number (RN) data queue **42**, an E/A command queue **44**, a PK command queue **46**, and data buffer(s) **47**.

Although **FIG. 1** illustrates an exemplary cryptographic data processing system architecture, it will be understood that the present invention is not limited to such a configuration, but is intended to encompass any configuration capable of carrying out operations described herein. Computer program code for carrying out operations of embodiments of the cryptographic data processing system **12** may be written in a high-level programming language, such as C or C++, for development

convenience. Nevertheless, some modules or routines may be written in assembly language or even micro-code to enhance performance and/or memory usage. It will be further appreciated that the functionality of any or all of the program modules may also be implemented using discrete hardware components, a single application specific integrated circuit (ASIC), or a programmed digital signal processor or microcontroller.

The present invention is described hereinafter with reference to flowchart and/or block diagram illustrations of methods, data processing systems, and/or computer program products in accordance with exemplary embodiments of the invention. It will be understood that each block of the flowchart and/or block diagram illustrations, and combinations of blocks in the flowchart and/or block diagram illustrations, may be implemented by computer program instructions. These computer program instructions may be provided to a processor of a general purpose computer, a special purpose computer, or other programmable data processing apparatus to produce a machine, such that the instructions, which execute via the processor of the computer or other programmable data processing apparatus, create means for implementing the functions/acts specified in the flowchart and/or block diagram block or blocks.

These computer program instructions may also be stored in a computer usable or computer-readable memory that may direct a computer or other programmable data processing apparatus to function in a particular manner, such that the instructions stored in the computer usable or computer-readable memory produce an article of manufacture including instructions that implement the function/act specified in the flowchart and/or block diagram block or blocks.

The computer program instructions may also be loaded onto a computer or other programmable data processing apparatus to cause a series of operational steps to be performed on the computer or other programmable apparatus to produce a computer implemented process such that the instructions that execute on the computer or other programmable apparatus provide steps for implementing the functions/acts specified in the flowchart and/or block diagram block or blocks.

Exemplary operations of cryptographic data processing systems, computer program products, and methods of operating same, in accordance with embodiments of the present invention, will be described hereafter. Referring now to **FIG. 2**, the host processor **16** loads a command block into one of the command queues **44** and **46**

at block **52**. The cryptographic accelerator processor **14** may be notified by the host processor **16** that the command block is available for processing or may periodically access the command queues **44**, and/or **46** to determine if a command block is available for processing. The cryptographic accelerator processor **14** downloads the command block from one of the command queues **44** and **46** and executes the command block at block **54**. Once the cryptographic accelerator processor **14** completes execution of the command block, the host processor **16** is notified at block **56**. Thus, according to embodiments of the present invention, the host processor **16** need not spend time interacting directly with the cryptographic accelerator processor **14** (*e.g.*, issuing a command to the cryptographic accelerator processor **14**, waiting for that command to complete, and then issuing another command). Instead, the host processor **16** may load commands into command queues **44** and **46**, which may then be processed in background by the cryptographic accelerator processor **14**. Moreover, the size and number of command block sequences may be less constrained because the availability of system memory is generally more abundant.

Referring now to **FIGS. 3 - 5**, the RNG execution unit **28**, the E/A execution unit **32**, and the PK engine execution unit **34** may use various registers that facilitate communication with the RN data queue **42** and the command queues **44** and **46**. For example, as shown in **FIG. 3**, a control/status register **62**, a RN data queue base address register **64**, a RN data queue size register **66**, and a RN data queue pointer register **68** may be defined for use by the RNG execution unit **28**. The control/status register **62** may include a self-test error field, which may be set if the RNG execution unit **28** generates two successive random number samples that are the same, and/or an error flag field, which may be used to notify the host processor **16** of an error on the system bus **26**. The RN data queue base address register **64** may be used to hold the base address of the RN data queue **42** in the system memory **22**. If the RN data queue **42** does not have a fixed size, then the RN data queue size register **66** may be used to hold the size of the RN data queue **42**. The RN data queue pointer register **68** may comprise a read pointer **72** portion and a write pointer **74** portion, which may be used by the RNG execution unit **28** and the host processor **16** as will be discussed in more detail hereinafter.

As shown in **FIG. 4**, a control/status register **82**, an E/A command queue base address register **84**, an E/A command queue size register **86**, and an E/A command queue pointer register **88** may be defined for use by the E/A execution unit **32**. The

control/status register **82** may include an interrupt flag field, which may be set if the host processor 16 requests an interrupt upon completion of a command block and/or if execution of a command block fails and/or an error flag field, which may be used to notify the host processor **16** of an error on the system bus **26**. The E/A command queue base address register **84** may be used to hold the base address of the E/A command queue **44** in the system memory **22**. If the E/A command queue **44** does not have a fixed size, then the E/A command queue size register **86** may be used to hold the size of the E/A command queue **44**. The E/A command queue pointer register **88** may comprise a read pointer **92** portion and a write pointer **94** portion, which may be used by the E/A execution unit **32** and the host processor **16**, respectively, as will be discussed in more detail hereinafter.

As shown in **FIG. 5**, a control/status register **102**, a PK command queue base address register **104**, a PK command queue size register **106**, and a PK command queue pointer register **108** may be defined for use by the PK engine execution unit **34**. The control/status register **102** may include an interrupt flag field, which may be set if the host processor **16** requests an interrupt upon completion of a command block and/or if execution of a command block fails and/or an error flag field, which may be used to notify the host processor **16** of an error on the system bus **26**. The PK command queue base address register **104** may be used to hold the base address of the PK command queue **46** in the system memory **22**. If the PK command queue **46** does not have a fixed size, then the PK command queue size register **106** may be used to hold the size of the PK command queue **46**. The PK command queue pointer register **108** may comprise a read pointer **112** portion and a write pointer **114** portion, which may be used by the PK engine execution unit **34** and the host processor **16**, respectively, as will be discussed in more detail hereinafter.

Referring now to **FIG. 6**, operations for loading a command block into the E/A command queue **44** and/or the PK command queue **46**, in accordance with embodiments of the present invention, will be described in more detail hereafter. In general, the host processor **16** writes commands into the command queues **44** and **46** beginning at write address locations stored in the write pointers for the respective command queues (*e.g.*, write pointers **94** and **114**). Before writing a command block into a command queue, however, the host processor determines at block **122** whether the write address plus the command block size equals the read address stored in the corresponding read pointer **92** or **112**. If the result determined at block **122** is "Yes,"

then the host processor **16** postpones loading a new command block into the command queue until the cryptographic accelerator processor **14** has incremented the read address. If, however, the result determined at block **122** is "No," then the host processor **16** loads a command block into the command queue at block **124** at the write address associated with the command queue and then increments the write address at block **126** by an amount corresponding to the size of the loaded command block. The host processor **16** need not check the current read address every time a new command block is loaded. Instead, the host processor **16** may check the read address when the write address is getting close to the last value the host processor **16** has for the read address. Checking the read address may be expensive in terms of processor cycles consumed. By checking the read address only when the read address is getting close to the write address (*e.g.*, within a predefined threshold), host processor 16 cycles may be conserved.

The foregoing operations are illustrated, for example, in **FIGS. 7** and **8**, which show embodiments of the E/A command queue **44** and the PK command queue **46**, respectively. As shown in **FIGS. 7** and **8**, both the E/A command queue **44** and the PK command queue **46** are configured to hold m command blocks, which each comprise eight, thirty-two bit words. The host processor **16** has written a single command block into the first command block position (*i.e.*, the "0" position) and the write address has been incremented to point to the next empty command block slot. The addresses used in **FIGS. 7** and **8** are based on command block slot numbers for purposes of illustration. These addresses may be converted into absolute addresses by multiplying the command block slot number by 256 and adding the resulting product to the respective base addresses for the command queues, which are stored in the E/A command queue base address register **84** and the PK command queue base address register **104**, respectively. Note that the test used at block **122** of **FIG. 6** to determine whether a new command block may be loaded into a command queue implies that if a command queue may hold up to m command blocks, then only m -1 command blocks may be stored in the command queue at the same time.

Referring now to **FIG. 9**, operations for executing a command block that has been loaded into the E/A command queue **44** and/or the PK command queue **46**, in accordance with embodiments of the present invention, will be described in more detail hereafter. At block **132**, the cryptographic accelerator processor **14** determines whether the write address is equal to the read address. Specifically, the E/A execution

unit **32** determines whether the write address is equal to the read address for the E/A command queue **44** and the PK engine execution unit **34** determines whether the write address is equal to the read address for the PK command queue **46**. If the result determined at block **132** is "Yes," then the cryptographic processor **14** waits until the host processor **16** loads a new command block into the command queue. If, however, the result determined at block **132** is "No," then the cryptographic accelerator processor **14** downloads the command block at the read address associated with the command queue and executes the command block at block **134**. In particular embodiments of the present invention, multiple command blocks may be downloaded for execution on the cryptographic accelerator processor **14** at the same time, which may further improve performance. The cryptographic accelerator processor **14** then increments the read address at block **136** by an amount corresponding to the size of the executed command block.

Returning to **FIGS. 7** and **8**, the read addresses are set to point to the first command block slot, which has been loaded with a command block by the host processor **16**. The E/A execution unit **32** and the PK engine execution unit **34** may read the command blocks loaded in the E/A command queue **44** and the PK command queue **46**, respectively, with only minimal interaction with the host processor **16**, *e.g.*, maintenance of the read pointers **92** and **112**, and the write pointers **94**, and **114**. In general, the cryptographic accelerator processor **14** may continue to execute commands located in a circular command queue in system memory until the read address equals the write address for that command queue.

In accordance with further embodiments of the present invention, interaction between the host processor **16** and the cryptographic accelerator processor **14** may be further reduced and overall system performance improved by including load and store commands in the cryptographic accelerator processor's command set. Referring now to **FIG. 10**, a load command loads one or more operands from the system memory **22** (*e.g.*, the data buffer(s) **47**) to the local memory **36** at block **142**. The cryptographic accelerator processor **14** then performs one or more operations on the operand(s) at block **144** to generate a result that is stored in the local memory **36**. A store command then stores the result in the system memory **22** at block **146**. Advantageously, the host processor **16** need not consume processing time downloading operands to the cryptographic accelerator processor **14** and/or uploading results from the cryptographic accelerator processor **14** into the system memory **22**.

To improve utilization of the chip area used to implement the cryptographic accelerator processor **14**, at least a portion of the operands downloaded from the system memory **22** may be stored in the local memory **36**. Instead of using a register number to identify the location of operands and results, an offset is used that identifies the relative position of the operands and results in the local memory **36**. For example, to perform the operation "a + b = c," a cryptographic accelerator processor **14** instruction may indicate that "a" is at offset 0 relative to a base address of the local memory **36**, "b" is at offset 8 relative to the base address of the local memory **36**, and the result "c" should be placed at offset 122 relative to the base address of the local memory **36**. In accordance with further embodiments of the present invention, the result generated in the local memory **36** may also be stored in a result field of a command block, which is located in one of the command queues **44** and **46** in the system memory **22**. Advantageously, operands and results may be packed together into the local memory **36**, which may conserve storage space. Because there is no wasted space in storing the operands and results in the local memory **36**, memory utilization may be improved. If the cryptographic accelerator processor **14** needs to be redesigned to handle larger operands, then the local memory **36** may be easier to resize than resizing several registers.

In accordance with further embodiments of the present invention, interaction between the host processor **16** and the cryptographic accelerator processor **14** may be further reduced and overall system performance improved by allowing the cryptographic accelerator processor **14** to inform the host processor **16** when command blocks have been executed. Referring now to **FIG. 11**, the host processor **16** loads a command block into one of the command queues **44** and **46** at block **152**. As shown in **FIG. 12A**, the command block may include an interrupt field, which may be set by the host processor **16** to turn an interrupt request on or off. The cryptographic accelerator processor **14** downloads the command block from one of the command queues **44** and **46** and executes the command block at block **154**. The cryptographic accelerator processor **14** may optionally store error information in the command block as shown in **FIG. 12B** at block **156**. The error information may comprise information that is associated with downloading the command block to the cryptographic accelerator processor **14** and/or executing the command block on the cryptographic accelerator processor **14**. At block **158**, if an interrupt has been requested in the interrupt field of the command block, then the cryptographic

accelerator processor **14** invokes an interrupt to notify the host processor **16** that the command block has completed.

In other embodiments of the present invention, instead of invoking an interrupt to notify the host processor **16** that a command block has been executed, the cryptographic accelerator processor **14** may update a completion field in the command block as shown in **FIG. 12C**. In addition, a periodic interrupt may be defined that upon each occurrence triggers the host processor **16** to check one or more of the command queues **44** and **46** to determine whether any of the command blocks stored therein have been executed by examining their completion fields. In still other embodiments of the present invention, the cryptographic accelerator processor **14** may store the results from executing a command block in the command block as shown in **FIG. 12D**.

In still other embodiments of the present invention, the host processor **16** may set a timer when storing a command block into a command queue **42**, **44**. Upon expiration of the timer, the host processor **16** may check to determine whether the command block has been executed. Advantageously, the status of a command block may be determined by the host processor **16** without the need to process an interrupt from the cryptographic accelerator processor **14**.

In accordance with further embodiments of the present invention, improved utilization of the system memory **22** may be attained by re-using at least a portion of a command block that contains input data to store a result or output that is generated by an adjunct processor, such as the cryptographic accelerator processor **14**, upon executing the command block. It is assumed that the size of the result or output is small enough to fit into the portion of the command block containing the input data that is to be overwritten. In addition, the region of the command block in which the result or output is stored should be selected carefully to ensure that the input data that is overwritten is no longer needed by the host processor **16** after the command block has been executed by the adjunct processor.

Referring now to **FIG. 13**, exemplary operations begin at block **162** where the host processor loads a command block that includes input data into one of the command queues **44** or **46** in the system memory **22**. Note that that instead of or in addition to including input data into the command block, the command block may include pointers to input data that reside, for example, in the data buffer(s) 47 in the system memory **22**. An adjunct processor, such as the cryptographic accelerator

processor **14**, may download the command block and perform one or more operations on the input data to generate a result at block **164**. If the command block includes pointers to input data, then the data are separately downloaded to the cryptographic accelerator processor **14** using the input data pointers. The result is then stored in the command block in the system memory **22** at block **166** such that at least a portion of the input data is overwritten. Advantageously, the memory reserved for the command block in the system memory **22** may be reduced because additional storage space need not be reserved to store the result of executing the command block either in the command block or elsewhere in the system memory **22**.

The foregoing operations are illustrated by way of example in **FIGS. 14A** and **14B**, which show an exemplary command block for decrypting an encrypted packet. Specifically, in **FIG. 14A**, a command block is shown that comprises a field that contains a hash key for the encrypted packet and another field that contains input information. The cryptographic accelerator processor **14** downloads the command block of **FIG. 14A** and performs hash operations using the hash key and input information to generate a hash value. As shown in **FIG. 14B**, this hash value is then stored in the command block in the system memory **22** by overwriting the input information, which is no longer needed once the hash value has been computed. Note that the input information may be one or more pointers to input data stored, for example, in the data buffer(s) **47** in the system memory **22**.

In accordance with further embodiments of the present invention illustrated in **FIG. 15**, the command block may include an input pointer field and/or an output pointer field, which are used to identify the location of the encrypted packet in the system memory **22** and the location where the decrypted packet is to be stored in the system memory **22**. For example, the cryptographic accelerator processor **14** may use the input pointer to download the encrypted packet from the system memory **22** and may then decrypt the encrypted packet using the hash key and input information to generate a hash value as discussed hereinabove. Note that the input information may be one or more pointers to input data stored, for example, in the data buffer(s) **47** in the system memory **22**. The hash value may be attached to the decrypted packet and the decrypted packet with the attached hash value may be stored in the system memory **22** at the address identified by the output pointer field in the command block.

Cryptographic processors and/or other types of signal processors and integrated circuits may use a hardware-based random number generator. The

cryptographic accelerator processor **14** may include a RNG execution unit **28** that may be used to generate random numbers for use by other execution units of the cryptographic accelerator processor **14** and/or the host processor **16**. Exemplary operations that may be used to reduce interaction between the host processor **16** and the cryptographic accelerator processor **14** and to improve overall system performance will be described hereafter. Referring now to **FIG. 16**, operations begin at block **172** where the cryptographic accelerator processor **14** loads a random number sample into the RN data queue **42** beginning at the write address stored in the write pointer field **74** of the RN data queue pointer register **68** (see **FIG. 3**). At block **174** the host processor **16** reads the random number sample in the RN data queue **42** beginning at the read address stored in the read pointer field **72** of the RN data queue pointer register **68** (see **FIG. 3**). Thus, according to embodiments of the present invention, the host processor **16** need not spend time interacting directly with the cryptographic accelerator processor **14** to request blocks of random data and/or reading random data from, for example, one or more registers on the cryptographic accelerator processor **14** chip.

Referring now to **FIG. 17**, operations for loading a random number sample into the RN data queue **42**, in accordance with embodiments of the present invention, will be described in more detail hereafter. Before writing a random number sample into the RN data queue **42**, the cryptographic accelerator processor **14** determines at block **182** whether the write address plus the random number sample size equals the read address stored in the read pointer field **72**. If the result determined at block **182** is "Yes," then the cryptographic processor **14** postpones loading a new random number sample into the RN data queue **42** until the host processor **16** has incremented the read address. If, however, the result determined at block **182** is "No," then the cryptographic processor **14** loads a random number sample into the RN data queue **42** at block **184** at the write address stored in the write pointer field **74** and then increments the write address at block **186** by an amount corresponding to the size of the loaded random number sample.

Note that in accordance with embodiments of the present invention, the cryptographic processor **14** may include a register and/or may recognize a command block that may be written to the cryptographic processor **14** that allows the host processor **16** to, for example, provide the cryptographic processor **14** with a random

number seed and/or instruct the cryptographic processor **14** to begin generating random numbers.

The foregoing operations are illustrated, for example, in **FIG. 18**, which shows an exemplary embodiment of the RN data queue **42**. As shown in **FIG. 18**, the RN data queue **42** is configured to hold 512 random number samples, which each comprise 64 bits. The cryptographic processor **14** has written four random number samples into addresses 1 through 4 and the write address has been incremented to point to the next available address, which is empty or contains data that have already been read by the host processor **16**. The addresses shown in **FIG. 18** are based on random number sample units for purposes of illustration. These addresses may be converted into absolute addresses by multiplying the random number sample number by 64 and adding the resulting product to the respective base address for the RN data queue **42**, which is stored in the RN data queue base address register **64**. Note that the test used at block **182** of **FIG. 17** to determine whether a new random number sample may be loaded into the RN data queue **42** implies that if the RN data queue **42** may hold up to m random number samples, then only m - 1 random number samples may be stored in the RN data queue **42** at the same time. Thus, if the RN data queue **42** is filled to its capacity, then it may hold 32,704 bits (511, 64-bit random number samples), which exceeds the 20,000 bits required by the Federal Information Processing Standard (FIPS) 140-1, Security Requirements for Cryptographic Modules issued January 11, 1994.

Referring now to **FIG. 19**, operations for reading a command block that has been loaded into the RN data queue **42**, in accordance with embodiments of the present invention, will be described in more detail hereafter. At block **192**, the host processor **16** determines whether the write address is equal to the read address. If the result determined at block **192** is "Yes," then the host processor **16** waits until the cryptographic accelerator processor **14** loads a new random number sample into the RN data queue **42**. If, however, the result determined at block **192** is "No," then the host processor **16** reads the random number sample at the read address stored in the read pointer field **72** at block **194**. The host processor **16** then increments the read address at block **196** by an amount corresponding to the size of the random number sample. The host processor **16** need not check the current write address every time a new random number sample is read. Instead, the host processor **16** may check the

write address when the read address is getting close to the last value the host processor **16** has for the write address.

Thus, according to embodiments of the present invention, a cryptographic accelerator processor **14** may provide random number samples for use by a host processor **16** with reduced interaction between the host processor **16** and the cryptographic accelerator processor **14**. In general, the host processor **14** need only interact with the cryptographic accelerator processor **14** to update the read address and to check the value of the write address when the read address approaches the last value the host processor **14** has for the write address. In addition, the cryptographic accelerator processor **14** may manage the buffering of the random number samples, which may conserve processor cycles of the host processor **16** and may reduce transactions on the system bus **26**, which may improve overall system performance.

The performance of cryptographic data processing systems may be affected by the system architecture and the methodology used to perform operations. For example, conventional cryptographic data processing systems may comprise one or more ASICs, such as the ASIC **202** shown in **FIG. 20**. The ASIC **202** comprises a plurality of functional units **204**, **206**, and **208**, which are configured to perform specific operations. As shown in **FIG. 20**, however, input commands are provided to the ASIC **202** serially and then routed to the appropriate functional unit **204**, **206**, and/or **208**. The outputs and/or results of executing the input commands are provided serially as command outputs from the ASIC **202**. Thus, the ASIC **202** typically processes commands sequentially such that a first command must finish before a subsequent command may be processed even if the commands are executed by different functional units.

Referring now to **FIG. 21**, the performance of cryptographic data processing systems may be improved, in accordance with embodiments of the present invention, by providing separate command interfaces that are respectively associated with the functional units such that each functional unit may receive command inputs and may generate command outputs and/or results independently of other functional units. As shown in **FIG. 21**, an ASIC **212** includes a plurality of functional units **214**, **216**, and **218**, which each receive command inputs through its own command interface and generate outputs and/or results that may be communicated to another processor through the command interface. By associating a separate command interface with each functional unit **214**, **216**, and **218**, the functional units **214**, **216**, and **218** may

operate independently and in parallel, thereby improving the performance of a cryptographic data processing system.

Referring now to **FIG. 22**, the functional units **214**, **216**, and **218** may comprise the E/A execution unit **32**, the RNG execution unit **28**, and the PK engine execution unit **34**. The E/A execution unit **32** comprises a command interface manager **222**, the RNG execution unit **28** comprises a command interface manager **224**, and the PK engine execution unit **34** comprises a command interface manager **226**. These respective command interface managers **222**, **224**, and **226** may be used to receive input command blocks from the E/A command queue **44**, to transmit random number samples to the RN data queue **42**, and to receive input command blocks from the PK command queue **46**, respectively, and to allow the respective execution units **28**, **32**, and **34** to perform operations in parallel.

Referring now to **FIG. 23**, operations of cryptographic data processing systems in which command interface managers are respectively associated with a plurality of functional units, in accordance with embodiments of the present invention, will be described hereafter. Operations begin at block **232** where one or more command blocks are provided to each of the functional units, such as, for example, by providing command blocks in the E/A command queue **44** and the PK command queue **46** for the E/A execution unit **32** and the PK engine execution unit **34**, respectively. At block **234**, the command blocks are simultaneously executed by the functional units by accessing the command blocks in parallel through, for example, the command interface manager **222** and the command interface manager **226**, which are associated with the E/A execution unit **32** and the PK engine execution unit **34**, respectively.

Note that command blocks may be provided to the cryptographic processor **14** in serial fashion over the system bus **24**. Nevertheless, the cryptographic processor 14 may distribute command blocks to the command interface managers **222**, **224**, and **226** associated with the execution units **32**, **28**, and **34**, which may then process the command blocks in parallel.

For purposes of illustration, exemplary embodiments of the present invention have been discussed hereinabove in which operations related to random number generation, encryption/authentication, and public key generation are performed in parallel based on functional units defined therefor. It will be understood that the operations that may be performed in parallel may be adjusted based on requirements

and/or needs. Moreover, commands may be provided to the command interface managers in a variety of ways. A processor may write commands directly to the command interface managers or, alternatively, commands may be stored in a memory and the command interface managers may be provided with the addresses where they may retrieve the stored commands for execution.

In summary, by performing operations in parallel using a plurality of functional units, the total number of operations that may be performed may be increased and the average latency for completing operations may be reduced.

The flowcharts of **FIGS. 2, 6, 9 - 11, 13, 16, 17, 19,** and **23** illustrate the architecture, functionality, and operations of possible embodiments of the cryptographic data processing system **12** of **FIG. 1**. In this regard, each block may represent a module, segment, or portion of code, which comprises one or more executable instructions for implementing the specified logical function(s). It should also be noted that in some alternative embodiments, the functions noted in the blocks may occur out of the order noted in **FIGS. 2, 6, 9 - 11, 13, 16, 17, 19,** and **23**. For example, two blocks shown in succession may in fact be executed substantially concurrently or the blocks may sometimes be executed in the reverse order, depending on the functionality involved.

In concluding the detailed description, it should be noted that many variations and modifications can be made to the preferred embodiments without substantially departing from the principles of the present invention. All such variations and modifications are intended to be included herein within the scope of the present invention, as set forth in the following claims.

CLAIMS

We claim:

1. A method of operating a cryptographic data processing system that comprises a host processor, a system memory coupled to the host processor, and a cryptographic processor integrated circuit that comprises a local memory and is coupled to the host processor and the system memory, the method comprising:

loading at least one operand from the system memory to the local memory;

performing at least one operation on the at least one operand to generate a result in the local memory; and

storing the result generated in the local memory in the system memory.

2. A method as recited in Claim 1, wherein performing the at least one operation, and storing the result are performed by the cryptographic processor without interaction with the host processor.

3. A cryptographic data processing system, comprising:

a host processor;

a system memory coupled to the host processor; and

a cryptographic processor that comprises a local memory and is coupled to the host processor and the system memory, the cryptographic processor being programmed to load at least one operand from the system memory to the local memory, perform at least one operation on the at least one operand to generate a result in the local memory, and store the result generated in the local memory in the system memory.

4. A method of operating a cryptographic data processing system that comprises a host processor, a system memory coupled to the host processor, and a cryptographic processor integrated circuit that comprises a local memory and is coupled to the host processor and the system memory, the method comprising:

providing a command queue in the system memory;

loading a command block into the command queue using the host processor;

executing the command block using the cryptographic processor; and

notifying the host processor that the command block has been executed.

5. A method as recited in Claim 4, further comprising:

providing a read address for the command queue and a write address for the command queue;

wherein loading the command block into the command queue using the host processor comprises loading the command block into the command queue using the host processor beginning at the write address, and wherein executing the command block using the cryptographic processor comprises executing the command block using the cryptographic processor beginning at the read address.

6. A method as recited in Claim 5, wherein loading the command block into the command queue using the host processor beginning at the write address comprises:

determining if the write address plus an amount corresponding to a size of a single command block equals the read address; and

loading the command block into the command queue using the host processor beginning at the write address if the write address plus the amount corresponding to the size of the single command block does not equal the read address.

7. A method as recited in Claim 6, further comprising:

incrementing the write address by the amount corresponding to the size of a single command block using the host processor after loading the command block into the command queue using the host processor beginning at the write address if the write address plus the amount corresponding to the size of the single command block does not equal the read address.

8. A method as recited in Claim 5, wherein executing the command block using the cryptographic processor beginning at the read address comprises:

determining whether the read address is equal to the write address; and

executing the command block using the cryptographic processor beginning at the read address if the read address is not equal to the write address.

9. A method as recited in Claim 8, further comprising:

incrementing the read address by an amount corresponding to a size of a single command block using the cryptographic processor after executing the command block using the cryptographic processor beginning at the read address.

10. A method as recited in Claim 4, wherein notifying the host processor that the command block has been executed comprises invoking an interrupt using the cryptographic processor after executing the command block.

11. A method as recited in Claim 4, wherein notifying the host processor that the command block has been executed comprises updating a completion field in the command block using the cryptographic processor.

12. A method as recited in Claim 11, further comprising:

providing a periodic interrupt; and

reading the completion field using the host processor upon invocation of the periodic interrupt.

13. A method as recited in Claim 4, wherein notifying the host processor that the command block has been executed comprises:

setting a timer after loading the command block into the command queue using the host processor; and

checking whether the command block has been executed after expiration of the timer.

14. A method as recited in Claim 4, further comprising:

loading at least one operand from the command queue to the local memory;

performing at least one operation on the at least one operand to generate a result in the local memory; and

storing the result generated in the local memory in the command queue.

15. A method of operating a cryptographic data processing system that comprises a host processor, a system memory coupled to the host processor, and a cryptographic processor integrated circuit that is coupled to the host processor and the system memory, the method comprising:

providing a command queue in the system memory;

loading a command block into the command queue using the host processor;

setting a value of an interrupt field in the command block to request an interrupt when the command block has been executed;

executing the command block using the cryptographic processor; and

invoking an interrupt using the cryptographic processor after executing the command block if the interrupt field in the command block is set to the value to request the interrupt.

16. A method as recited in Claim 15, further comprising:

storing error information in the command block that is associated with executing the command block using the cryptographic processor.

17. A method of operating a cryptographic data processing system that comprises a host processor, a system memory coupled to the host processor, and a cryptographic processor integrated circuit that is coupled to the host processor and the system memory, the method comprising:

providing a command queue in the system memory;

loading a command block into the command queue using the host processor;

executing the command block using the cryptographic processor; and

updating a completion field in the command block using the cryptographic processor.

18. A method as recited in Claim 17, further comprising:

providing a periodic interrupt; and

reading the completion field using the host processor upon invocation of the periodic interrupt.

19. A method as recited in Claim 17, further comprising:

storing error information in the command block that is associated with executing the command block using the cryptographic processor.

20. A method of operating a data processing system that comprises a host processor, a system memory coupled to the host processor, and an adjunct processor

integrated circuit that is coupled to the host processor and the system memory, the method comprising:

providing a command queue in the system memory;

loading a command block into the command queue using the host processor, the command block comprising an input data field that contains input data;

performing an operation based on the input data using the adjunct processor to generate a result; and

storing the result in the input data field such that at least a portion of the input data is overwritten.

21. A method as recited in Claim 20, wherein the data processing system comprises a cryptographic data processing system, the adjunct processor integrated circuit comprises a cryptographic processor integrated circuit, and performing the operation based on the input data comprises:

performing a hash operation based on the input data using the cryptographic processor to generate a hash value.

22. A method as recited in Claim 21, wherein storing the result in the input data field comprises:

storing the hash value in the input data field such that the at least a portion of the input data is overwritten.

23. A method as recited in Claim 21, wherein the command block further comprises an input pointer field that contains an address in the system memory of an incoming packet and wherein performing the hash operation comprises:

performing the hash operation based on the input data and the incoming packet using the cryptographic processor to generate the hash value.

24. A method as recited in Claim 23, wherein the command block further comprises an output pointer field that contains an address in the system memory for storing a decrypted packet, the method further comprising:

decrypting the incoming packet using the cryptographic processor to generate the decrypted packet;

attaching the hash value to the decrypted packet; and

storing the decrypted packet with the attached hash value at the address in the system memory contained in the output pointer field.

25. A method of operating a cryptographic data processing system that comprises a host processor, a system memory coupled to the host processor, and a cryptographic processor integrated circuit that comprises a local memory and is coupled to the host processor and the system memory, the method comprising:

providing a command queue in the system memory;

providing a read address for the command queue and a write address for the command queue;

loading a random number sample into the command queue using the cryptographic processor beginning at the write address; and

reading the random number sample using the host processor beginning at the read address.

26. A method as recited in Claim 25, wherein loading the random number sample into the command queue using the cryptographic processor beginning at the write address comprises:

determining if the write address plus an amount corresponding to a size of a single random number sample equals the read address; and

loading the random number sample into the command queue using the cryptographic processor beginning at the write address if the write address plus the amount corresponding to the size of the single random number sample does not equal the read address.

27. A method as recited in Claim 26, further comprising:

incrementing the write address by the amount corresponding to the size of a single random number sample using the cryptographic processor after loading the random number sample into the command queue using the cryptographic processor beginning at the write address if the write address plus the amount corresponding to the size of the single random number sample does not equal the read address.

28. A method as recited in Claim 25, wherein reading the random number sample using the host processor beginning at the read address comprises:

determining whether the read address is equal to the write address; and

reading the random number sample using the host processor beginning at the read address if the read address is not equal to the write address.

29. A method as recited in Claim 28, further comprising:

incrementing the read address by an amount corresponding to a size of a single random number sample using the host processor after reading the random number sample using the host processor beginning at the read address.

30. A method of operating a data processing system that comprises a host processor, a system memory coupled to the host processor, and an adjunct processor integrated circuit that is coupled to the host processor and the system memory, the method comprising:

transferring information between the host processor and the adjunct processor using the system memory.

31. A cryptographic data processing system that comprises a host processor, a system memory coupled to the host processor, and a cryptographic processor integrated circuit that comprises a local memory and is coupled to the host processor and the system memory, the system further comprising:

means for loading at least one operand from the system memory to the local memory;

means for performing at least one operation on the at least one operand to generate a result in the local memory; and

means for storing the result generated in the local memory in the system memory.

32. A cryptographic data processing system as recited in Claim 31, wherein the means for performing the at least one operation, and the means for storing the result execute without interaction with the host processor.

33. A cryptographic data processing system that comprises a host processor, a system memory coupled to the host processor, and a cryptographic

processor integrated circuit that comprises a local memory and is coupled to the host processor and the system memory, the system further comprising:

means for providing a command queue in the system memory;

means for loading a command block into the command queue using the host processor;

means for executing the command block using the cryptographic processor; and

means for notifying the host processor that the command block has been executed.

34. A cryptographic data processing system as recited in Claim 33, further comprising:

means for providing a read address for the command queue and a write address for the command queue;

wherein the means for loading the command block into the command queue using the host processor comprises means for loading the command block into the command queue using the host processor beginning at the write address, and wherein the means for executing the command block using the cryptographic processor comprises means for executing the command block using the cryptographic processor beginning at the read address.

35. A cryptographic data processing system as recited in Claim 34, wherein the means for loading the command block into the command queue using the host processor beginning at the write address comprises:

means for determining if the write address plus an amount corresponding to a size of a single command block equals the read address; and

means for loading the command block into the command queue using the host processor beginning at the write address if the write address plus the amount corresponding to the size of the single command block does not equal the read address.

36. A cryptographic data processing system as recited in Claim 35, further comprising:

means for incrementing the write address by the amount corresponding to the size of a single command block using the host processor if the write address plus the amount corresponding to the size of the single command block does not equal the read address, the means for incrementing being responsive to the means for loading the command block into the command queue using the host processor beginning at the write address.

37. A cryptographic data processing system as recited in Claim 34, wherein the means for executing the command block using the cryptographic processor beginning at the read address comprises:

means for determining whether the read address is equal to the write address; and

means for executing the command block using the cryptographic processor beginning at the read address if the read address is not equal to the write address.

38. A cryptographic data processing system as recited in Claim 37, further comprising:

means for incrementing the read address by an amount corresponding to a size of a single command block using the cryptographic processor, the means for incrementing being responsive to the means for executing the command block using the cryptographic processor beginning at the read address.

39. A cryptographic data processing system as recited in Claim 33, wherein the means for notifying the host processor that the command block has been executed comprises means for invoking an interrupt using the cryptographic processor after executing the command block.

40. A cryptographic data processing system as recited in Claim 33, wherein the means for notifying the host processor that the command block has been executed comprises means for updating a completion field in the command block using the cryptographic processor.

41. A cryptographic data processing system as recited in Claim 40, further comprising:

means for providing a periodic interrupt; and

means for reading the completion field using the host processor upon invocation of the periodic interrupt.

42. A cryptographic data processing system as recited in Claim 33, wherein the means for notifying the host processor that the command block has been executed comprises:

means for setting a timer after loading the command block into the command queue using the host processor; and

means for checking whether the command block has been executed after expiration of the timer.

43. A cryptographic data processing system as recited in Claim 33, further comprising:

means for loading at least one operand from the command queue to the local memory;

means for performing at least one operation on the at least one operand to generate a result in the local memory; and

means for storing the result generated in the local memory in the command queue.

44. A cryptographic data processing system that comprises a host processor, a system memory coupled to the host processor, and a cryptographic processor integrated circuit that is coupled to the host processor and the system memory, the system further comprising:

means for providing a command queue in the system memory;

means for loading a command block into the command queue using the host processor;

means for setting a value of an interrupt field in the command block to request an interrupt when the command block has been executed;

means for executing the command block using the cryptographic processor; and

means for invoking an interrupt using the cryptographic processor after executing the command block if the interrupt field in the command block is set to the value to request the interrupt.

45. A cryptographic data processing system as recited in Claim 44, further comprising:

means for storing error information in the command block that is associated with executing the command block using the cryptographic processor.

46. A cryptographic data processing system that comprises a host processor, a system memory coupled to the host processor, and a cryptographic processor integrated circuit that is coupled to the host processor and the system memory, the system further comprising:

means for providing a command queue in the system memory;

means for loading a command block into the command queue using the host processor;

means for executing the command block using the cryptographic processor; and

means for updating a completion field in the command block using the cryptographic processor.

47. A cryptographic data processing system as recited in Claim 46, further comprising:

means for providing a periodic interrupt; and

means for reading the completion field using the host processor upon invocation of the periodic interrupt.

48. A cryptographic data processing system as recited in Claim 46, further comprising:

means for storing error information in the command block that is associated with executing the command block using the cryptographic processor.

49. A data processing system that comprises a host processor, a system memory coupled to the host processor, and an adjunct processor integrated circuit that

is coupled to the host processor and the system memory, the system further comprising:

means for providing a command queue in the system memory;

means for loading a command block into the command queue using the host processor, the command block comprising an input data field that contains input data;

means for performing an operation based on the input data using the adjunct processor to generate a result; and

means for storing the result in the input data field such that at least a portion of the input data is overwritten.

50. A data processing system as recited in Claim 49, wherein the data processing system comprises a cryptographic data processing system, the adjunct processor integrated circuit comprises a cryptographic processor integrated circuit, and the means for performing the operation based on the input data comprises:

means for performing a hash operation based on the input data using the cryptographic processor to generate a hash value.

51. A data processing system as recited in Claim 50, wherein the means for storing the result in the input data field comprises:

means for storing the hash value in the input data field such that the at least a portion of the input data is overwritten.

52. A data processing system as recited in Claim 50, wherein the command block further comprises an input pointer field that contains an address in the system memory of an incoming packet and wherein the means for performing the hash operation comprises:

means for performing the hash operation based on the input data and the incoming packet using the cryptographic processor to generate the hash value.

53. A data processing system as recited in Claim 52, wherein the command block further comprises an output pointer field that contains an address in the system memory for storing a decrypted packet, the data processing system further comprising:

means for decrypting the incoming packet using the cryptographic processor to generate the decrypted packet;

means for attaching the hash value to the decrypted packet; and

means for storing the decrypted packet with the attached hash value at the address in the system memory contained in the output pointer field.

54. A cryptographic data processing system that comprises a host processor, a system memory coupled to the host processor, and a cryptographic processor integrated circuit that comprises a local memory and is coupled to the host processor and the system memory, the system further comprising:

means for providing a command queue in the system memory;

means for providing a read address for the command queue and a write address for the command queue;

means for loading a random number sample into the command queue using the cryptographic processor beginning at the write address; and

means for reading the random number sample using the host processor beginning at the read address.

55. A cryptographic data processing system as recited in Claim 54, wherein the means for loading the random number sample into the command queue using the cryptographic processor beginning at the write address comprises:

means for determining if the write address plus an amount corresponding to a size of a single random number sample equals the read address; and

means for loading the random number sample into the command queue using the cryptographic processor beginning at the write address if the write address plus the amount corresponding to the size of the single random number sample does not equal the read address.

56. A cryptographic data processing system as recited in Claim 55, further comprising:

means for incrementing the write address by the amount corresponding to the size of a single random number sample using the cryptographic processor if the write address plus the amount corresponding to the size of the single random number sample does not equal the read address, the means for incrementing being responsive

to the means for loading the random number sample into the command queue using the cryptographic processor beginning at the write address.

57. A cryptographic data processing system as recited in Claim 54, wherein the means for reading the random number sample using the host processor beginning at the read address comprises:

means for determining whether the read address is equal to the write address; and

means for reading the random number sample using the host processor beginning at the read address if the read address is not equal to the write address.

58. A cryptographic data processing system as recited in Claim 57, further comprising:

means for incrementing the read address by an amount corresponding to a size of a single random number sample using the host processor, the means for incrementing being responsive to the means for reading the random number sample using the host processor beginning at the read address.

59. A computer program product for operating a cryptographic data processing system that comprises a host processor, a system memory coupled to the host processor, and a cryptographic processor integrated circuit that comprises a local memory and is coupled to the host processor and the system memory, the computer program product comprising:

a computer readable program medium having computer readable program code embodied therein, the computer readable program code comprising:

computer readable program code for loading at least one operand from the system memory to the local memory;

computer readable program code for performing at least one operation on the at least one operand to generate a result in the local memory; and

computer readable program code for storing the result generated in the local memory in the system memory.

60. A computer program product as recited in Claim 59, wherein the computer readable program code for performing the at least one operation, and the

computer readable program code for storing the result execute without interaction with the host processor.

61. A computer program product for operating a cryptographic data processing system that comprises a host processor, a system memory coupled to the host processor, and a cryptographic processor integrated circuit that comprises a local memory and is coupled to the host processor and the system memory, the computer program product comprising:

a computer readable program medium having computer readable program code embodied therein, the computer readable program code comprising:

computer readable program code for providing a command queue in the system memory;

computer readable program code for loading a command block into the command queue using the host processor;

computer readable program code for executing the command block using the cryptographic processor; and

computer readable program code for notifying the host processor that the command block has been executed.

62. A computer program product as recited in Claim 61, further comprising:

computer readable program code for providing a read address for the command queue and a write address for the command queue;

wherein the computer readable program code for loading the command block into the command queue using the host processor comprises computer readable program code for loading the command block into the command queue using the host processor beginning at the write address, and wherein the computer readable program code for executing the command block using the cryptographic processor comprises computer readable program code for executing the command block using the cryptographic processor beginning at the read address.

63. A computer program product as recited in Claim 62, wherein the computer readable program code for loading the command block into the command queue using the host processor beginning at the write address comprises:

computer readable program code for determining if the write address plus an amount corresponding to a size of a single command block equals the read address; and

computer readable program code for loading the command block into the command queue using the host processor beginning at the write address if the write address plus the amount corresponding to the size of the single command block does not equal the read address.

64. A computer program product as recited in Claim 63, further comprising:

computer readable program code for incrementing the write address by the amount corresponding to the size of a single command block using the host processor if the write address plus the amount corresponding to the size of the single command block does not equal the read address, the computer readable program code for incrementing being responsive to the computer readable program code for loading the command block into the command queue using the host processor beginning at the write address.

65. A computer program product as recited in Claim 62, wherein the computer readable program code for executing the command block using the cryptographic processor beginning at the read address comprises:

computer readable program code for determining whether the read address is equal to the write address; and

computer readable program code for executing the command block using the cryptographic processor beginning at the read address if the read address is not equal to the write address.

66. A computer program product as recited in Claim 65, further comprising:

computer readable program code for incrementing the read address by an amount corresponding to a size of a single command block using the cryptographic processor, the computer readable program code for incrementing being responsive to the computer readable program code for executing the command block using the cryptographic processor beginning at the read address.

67. A computer program product as recited in Claim 61, wherein the computer readable program code for notifying the host processor that the command block has been executed comprises computer readable program code for invoking an interrupt using the cryptographic processor after executing the command block.

68. A computer program product as recited in Claim 61, wherein the computer readable program code for notifying the host processor that the command block has been executed comprises computer readable program code for updating a completion field in the command block using the cryptographic processor.

69. A computer program product as recited in Claim 68, further comprising:

computer readable program code for providing a periodic interrupt; and

computer readable program code for reading the completion field using the host processor upon invocation of the periodic interrupt.

70. A method as recited in Claim 61, wherein the computer readable program code for notifying the host processor that the command block has been executed comprises:

computer readable program code for setting a timer after loading the command block into the command queue using the host processor; and

computer readable program code for checking whether the command block has been executed after expiration of the timer.

71. A computer program product as recited in Claim 61, further comprising:

computer readable program code for loading at least one operand from the command queue to the local memory;

computer readable program code for performing at least one operation on the at least one operand to generate a result in the local memory; and

computer readable program code for storing the result generated in the local memory in the command queue.

72. A computer program product for operating a cryptographic data processing system that comprises a host processor, a system memory coupled to the host processor, and a cryptographic processor integrated circuit that is coupled to the host processor and the system memory, the computer program product comprising:

a computer readable program medium having computer readable program code embodied therein, the computer readable program code comprising:

computer readable program code for providing a command queue in the system memory;

computer readable program code for loading a command block into the command queue using the host processor;

computer readable program code for setting a value of an interrupt field in the command block to request an interrupt when the command block has been executed;

computer readable program code for executing the command block using the cryptographic processor; and

computer readable program code for invoking an interrupt using the cryptographic processor after executing the command block if the interrupt field in the command block is set to the value to request the interrupt.

73. A computer program product as recited in Claim 72, further comprising:

computer readable program code for storing error information in the command block that is associated with executing the command block using the cryptographic processor.

74. A computer program product for operating a cryptographic data processing system that comprises a host processor, a system memory coupled to the host processor, and a cryptographic processor integrated circuit that is coupled to the host processor and the system memory, the computer program product comprising:

a computer readable program medium having computer readable program code embodied therein, the computer readable program code comprising:

computer readable program code for providing a command queue in the system memory;

computer readable program code for loading a command block into the command queue using the host processor;

computer readable program code for executing the command block using the cryptographic processor; and

computer readable program code for updating a completion field in the command block using the cryptographic processor.

75. A computer program product as recited in Claim 74, further comprising:

computer readable program code for providing a periodic interrupt; and

computer readable program code for reading the completion field using the host processor upon invocation of the periodic interrupt.

76. A computer program product as recited in Claim 74, further comprising:

computer readable program code for storing error information in the command block that is associated with executing the command block using the cryptographic processor.

77. A computer program product for operating a data processing system that comprises a host processor, a system memory coupled to the host processor, and an adjunct processor integrated circuit that is coupled to the host processor and the system memory, the computer program product comprising:

a computer readable program medium having computer readable program code embodied therein, the computer readable program code comprising:

computer readable program code for providing a command queue in the system memory;

computer readable program code for loading a command block into the command queue using the host processor, the command block comprising an input data field that contains input data;

computer readable program code for performing an operation based on the input data using the adjunct processor to generate a result; and

computer readable program code for storing the result in the input data field such that at least a portion of the input data is overwritten.

78. A computer program product as recited in Claim 77, wherein the data processing system comprises a cryptographic data processing system, the adjunct processor integrated circuit comprises a cryptographic processor integrated circuit, and the computer readable program code for performing the operation based on the input data comprises:

computer readable program code for performing a hash operation based on the input data using the cryptographic processor to generate a hash value.

79. A computer program product as recited in Claim 78, wherein the computer readable program code for storing the result in the input data field comprises:

computer readable program code for storing the hash value in the input data field such that the at least a portion of the input data is overwritten.

80. A computer program product as recited in Claim 78, wherein the command block further comprises an input pointer field that contains an address in the system memory of an incoming packet and wherein the computer readable program code for performing the hash operation comprises:

computer readable program code for performing the hash operation based on the input data and the incoming packet using the cryptographic processor to generate the hash value.

81. A computer program product as recited in Claim 80, wherein the command block further comprises an output pointer field that contains an address in the system memory for storing a decrypted packet, the computer program product further comprising:

computer readable program code for decrypting the incoming packet using the cryptographic processor to generate the decrypted packet;

computer readable program code for attaching the hash value to the decrypted packet; and

computer readable program code for storing the decrypted packet with the attached hash value at the address in the system memory contained in the output pointer field.

82. A computer program product for operating cryptographic data processing system that comprises a host processor, a system memory coupled to the host processor, and a cryptographic processor integrated circuit that comprises a local memory and is coupled to the host processor and the system memory, the computer program product comprising:

a computer readable program medium having computer readable program code embodied therein, the computer readable program code comprising:

computer readable program code for providing a command queue in the system memory;

computer readable program code for providing a read address for the command queue and a write address for the command queue;

computer readable program code for loading a random number sample into the command queue using the cryptographic processor beginning at the write address; and

computer readable program code for reading the random number sample using the host processor beginning at the read address.

83. A computer program product as recited in Claim 82, wherein the computer readable program code for loading the random number sample into the command queue using the cryptographic processor beginning at the write address comprises:

computer readable program code for determining if the write address plus an amount corresponding to a size of a single random number sample equals the read address; and

computer readable program code for loading the random number sample into the command queue using the cryptographic processor beginning at the write address if the write address plus the amount corresponding to the size of the single random number sample does not equal the read address.

84. A computer program product as recited in Claim 83, further comprising:

computer readable program code for incrementing the write address by the amount corresponding to the size of a single random number sample using the cryptographic processor if the write address plus the amount corresponding to the size of the single random number sample does not equal the read address, the computer

readable program code for incrementing being responsive to the computer readable program code for loading the random number sample into the command queue using the cryptographic processor beginning at the write address.

85. A computer program product as recited in Claim 82, wherein the computer readable program code for reading the random number sample using the host processor beginning at the read address comprises:

computer readable program code for determining whether the read address is equal to the write address; and

computer readable program code for reading the random number sample using the host processor beginning at the read address if the read address is not equal to the write address.

86. A computer program product as recited in Claim 85, further comprising:

computer readable program code for incrementing the read address by an amount corresponding to a size of a single random number sample using the host processor, the computer readable program code for incrementing being responsive to the computer readable program code for reading the random number sample using the host processor beginning at the read address.

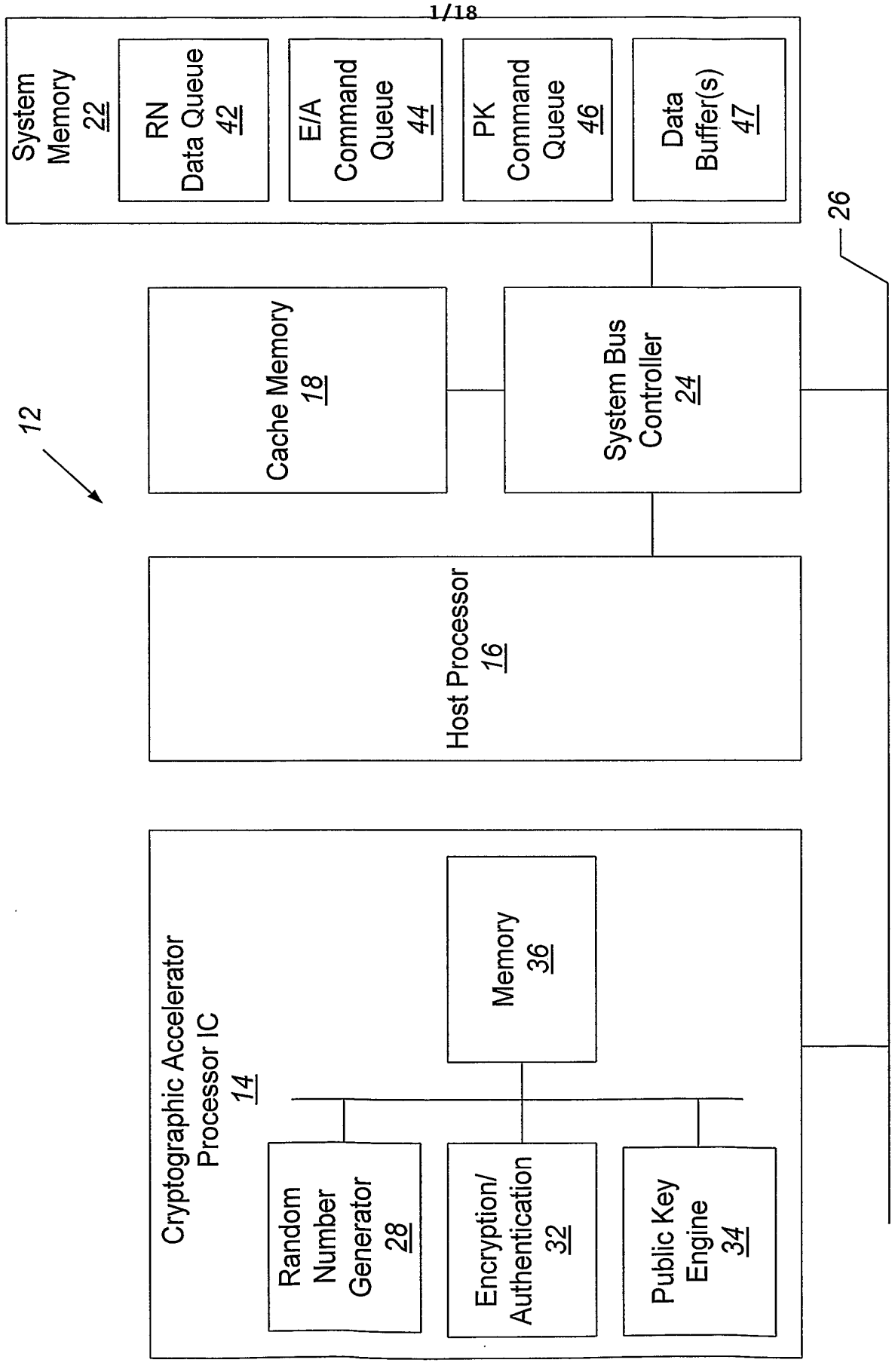
12
System Memory 22
RN Data Queue 42
E/A Command Queue 44
PK Command Queue 46
Data Buffer(s) 47
26
Cache Memory 18
System Bus Controller 24
Host Processor 16
Cryptographic Accelerator Processor IC 14
Memory 36
Random Number Generator 28
Encryption/ Authentication 32
Public Key Engine 34

FIG. 1

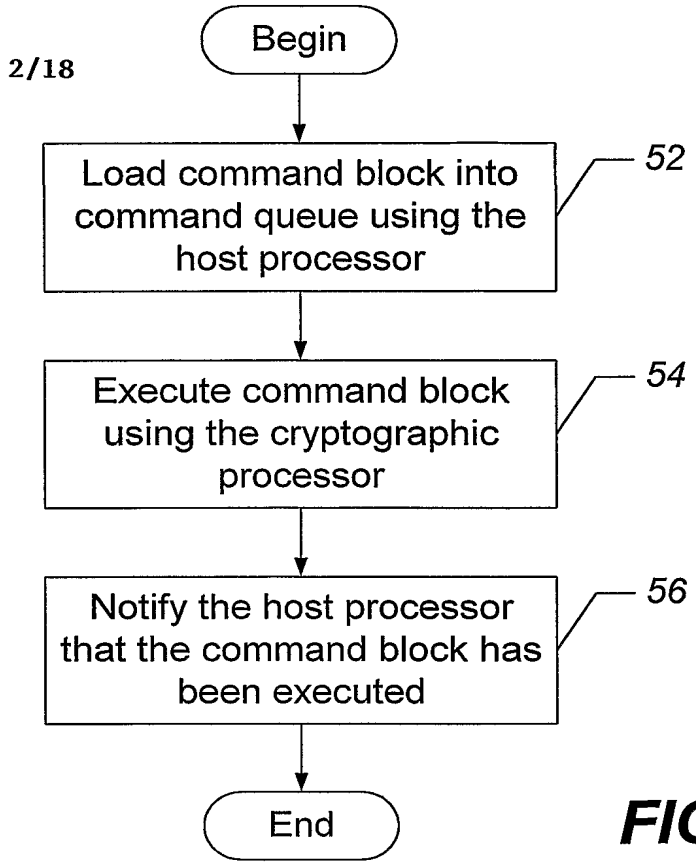
Begin
Load command block into command queue using the host processor
52
Execute command block using the cryptographic processor
54
Notify the host processor that the command block has been executed
56
End

FIG. 2

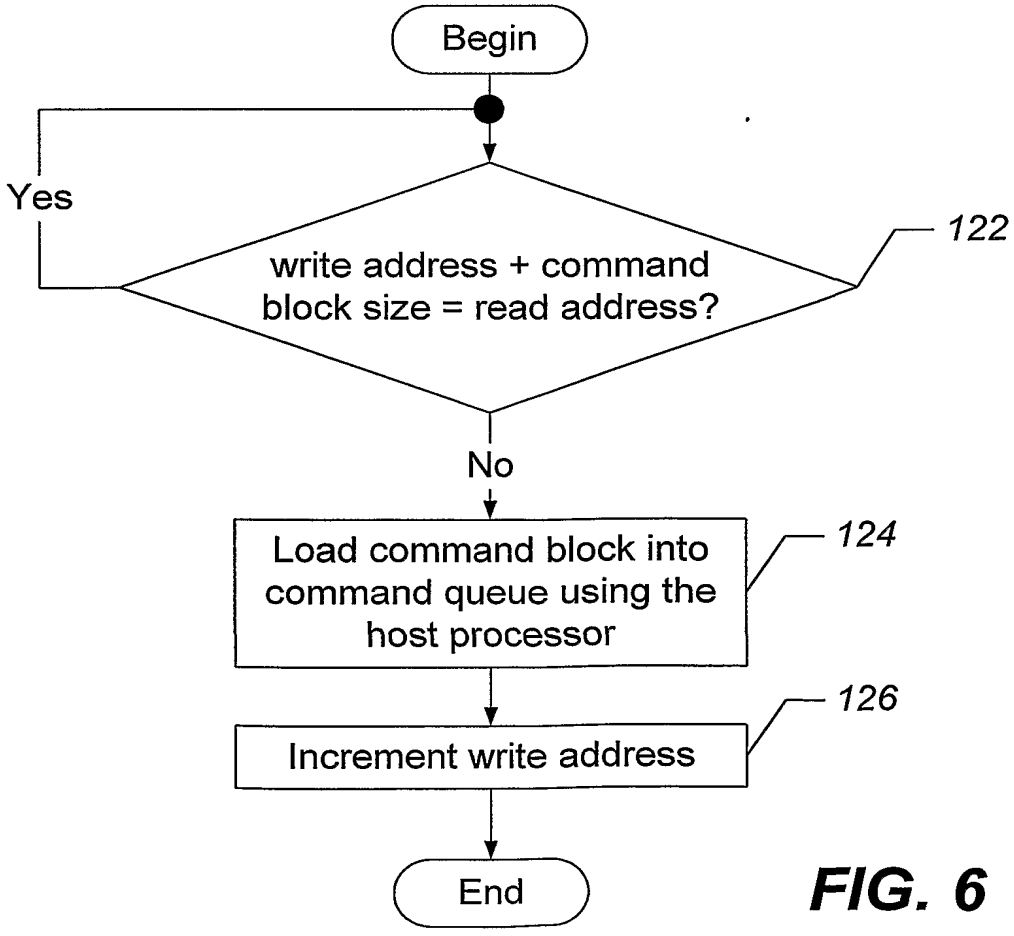
Begin
Yes
write address + command block size = read address?
122
No
Load command block into command queue using the host processor
124
Increment write address
126
End

FIG. 6

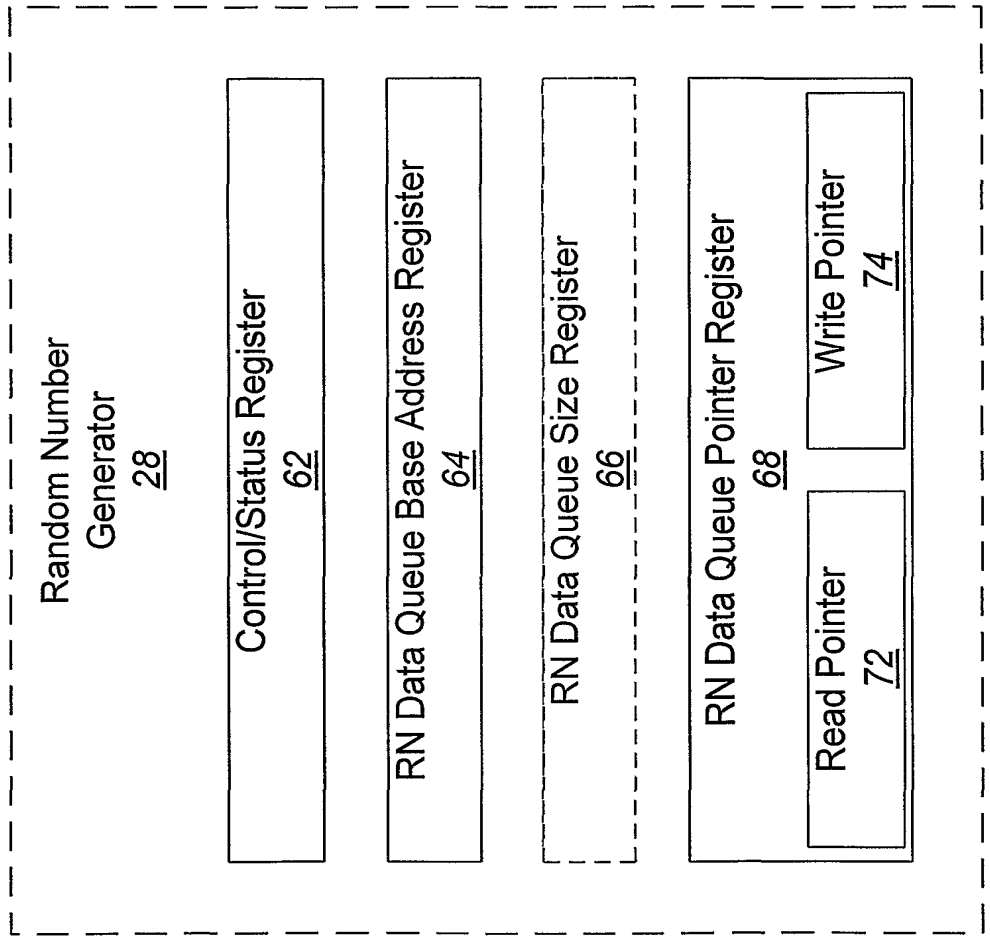
Random Number Generator
28
Control/Status Register
62
RN Data Queue Base Address Register
64
RN Data Queue Size Register
66
RN Data Queue Pointer Register
68
Read Pointer
72
Write Pointer
74

FIG. 3

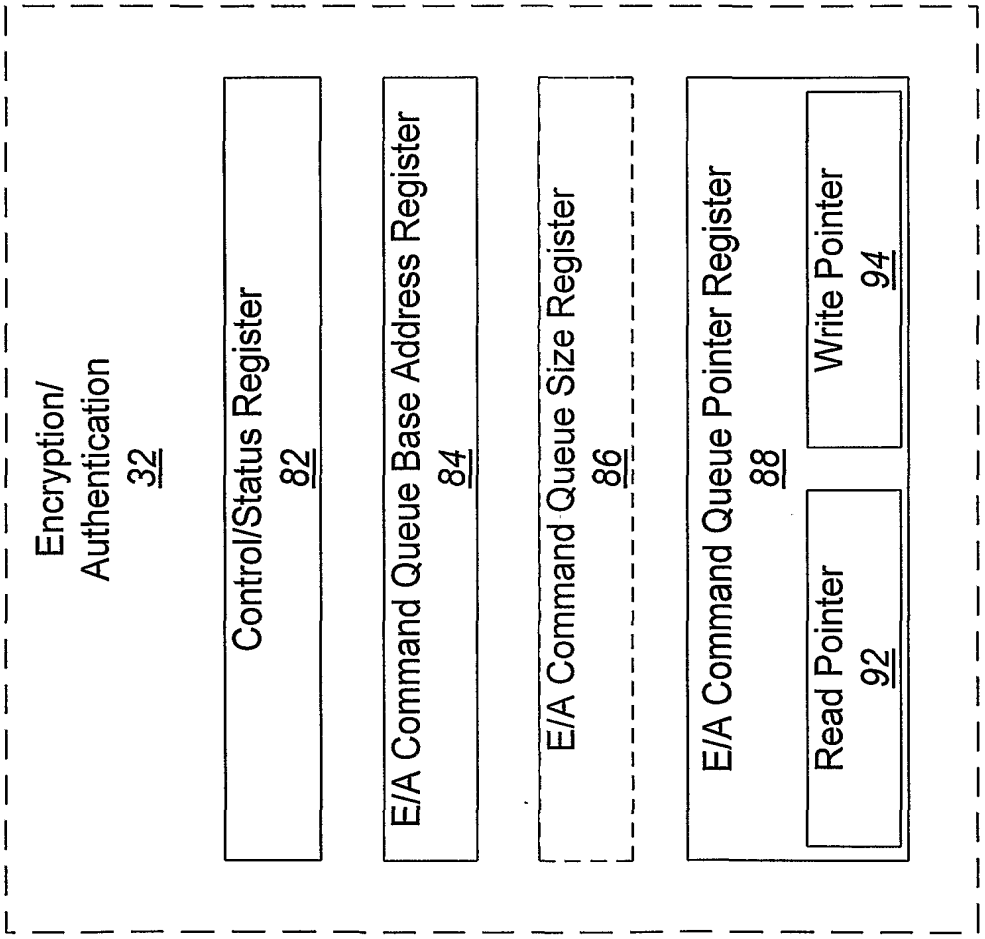
Encryption/
Authentication
32
Control/Status Register
82
E/A Command Queue Base Address Register
84
E/A Command Queue Size Register
86
E/A Command Queue Pointer Register
88
Read Pointer
92
Write Pointer
94

FIG. 4

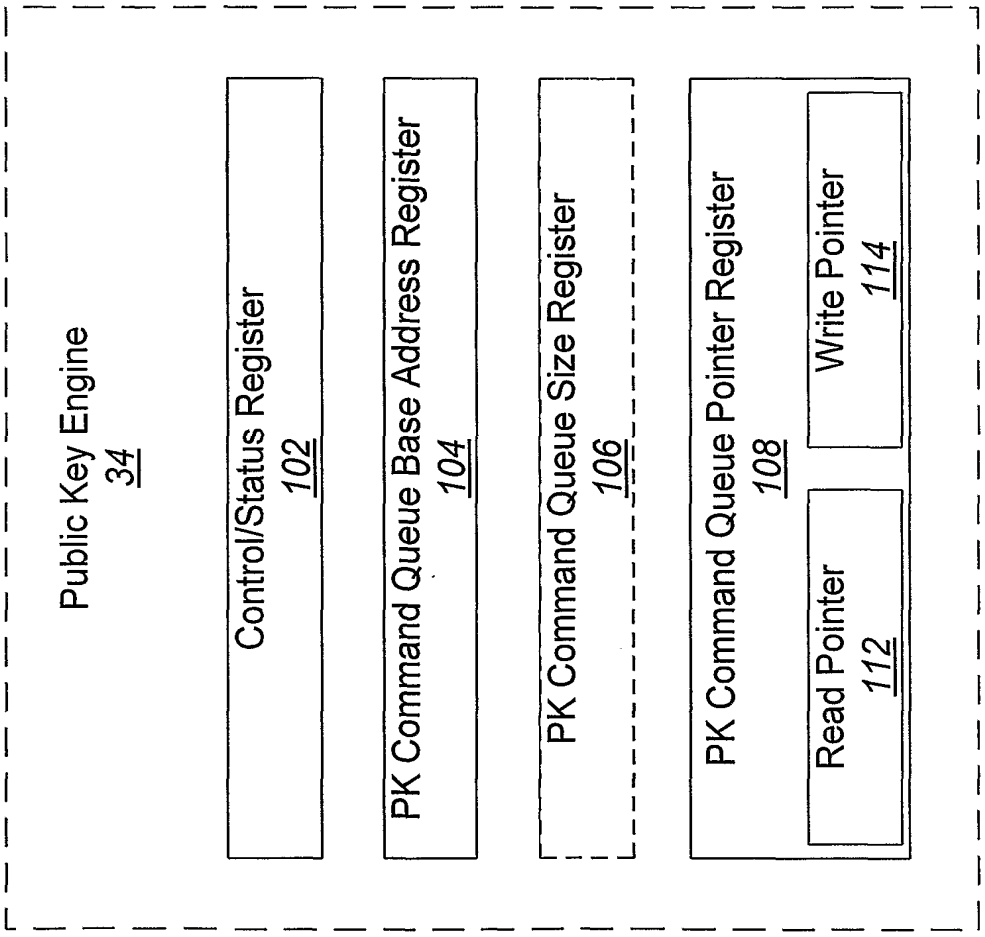
Public Key Engine
34
Control/Status Register
102
PK Command Queue Base Address Register
104
PK Command Queue Size Register
106
PK Command Queue Pointer Register
108
Read Pointer
112
Write Pointer
114

FIG. 5

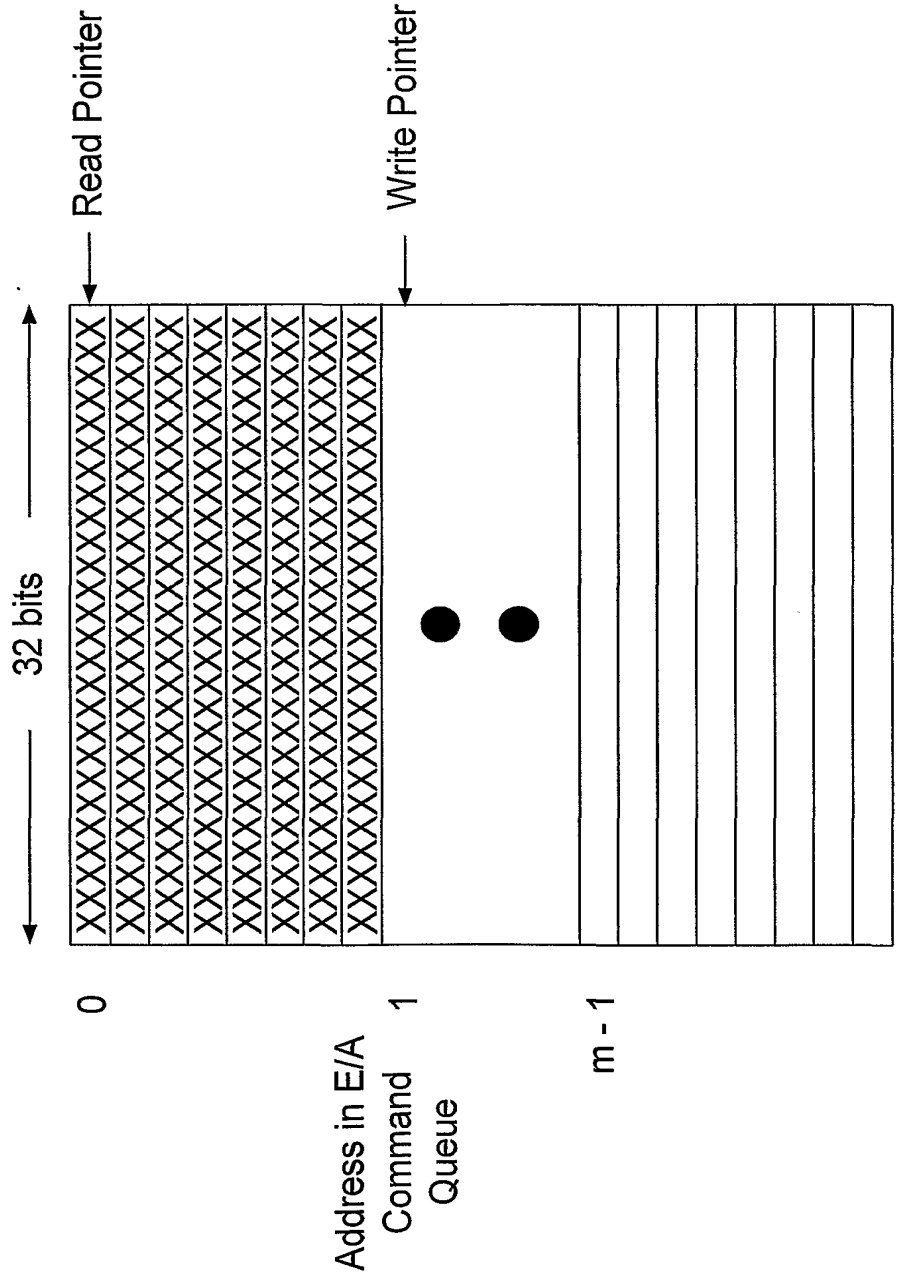
Read Pointer
Write Pointer
32 bits
0
1
m - 1
Address in E/A Command Queue

FIG. 7

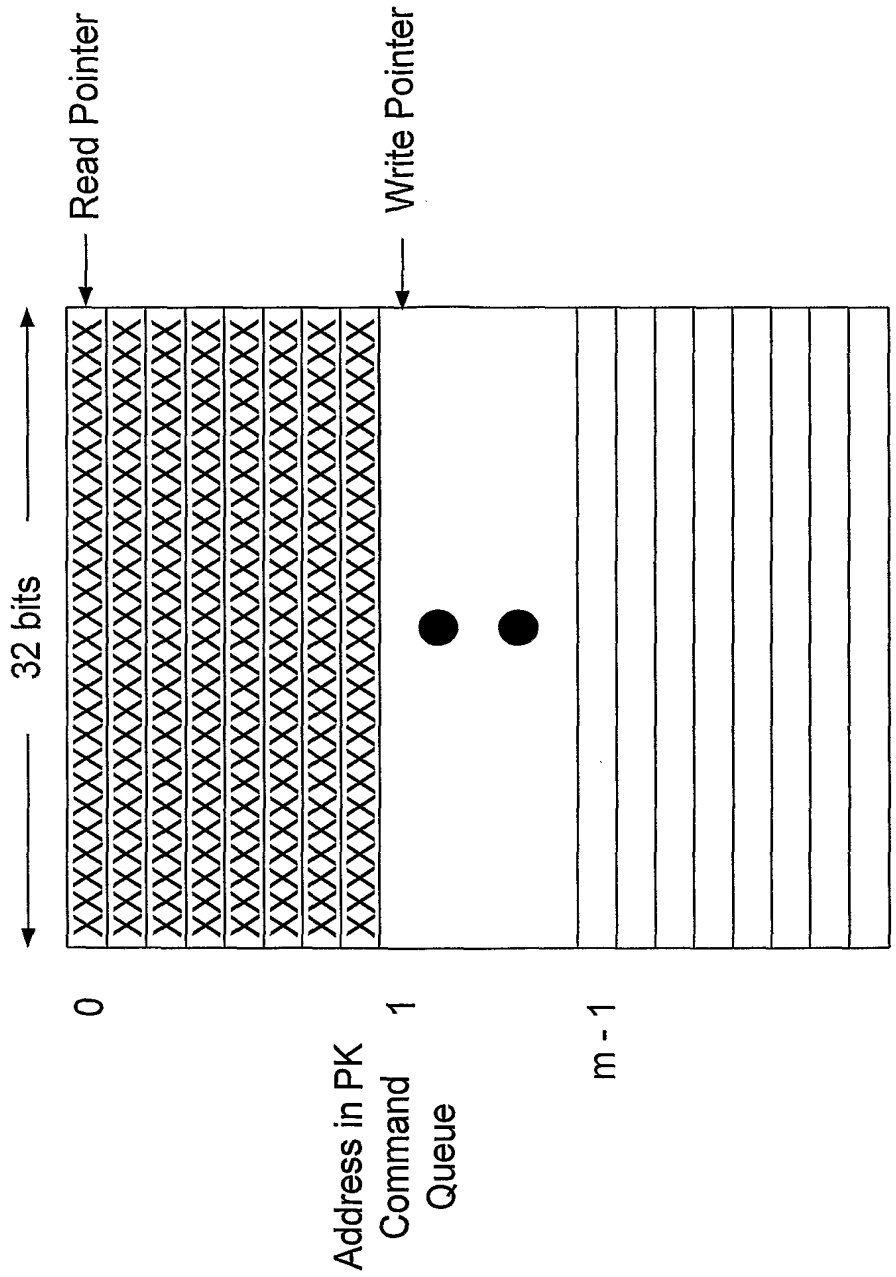
Read Pointer
Write Pointer
32 bits
0
1
m - 1
Address in PK
Command
Queue

FIG. 8

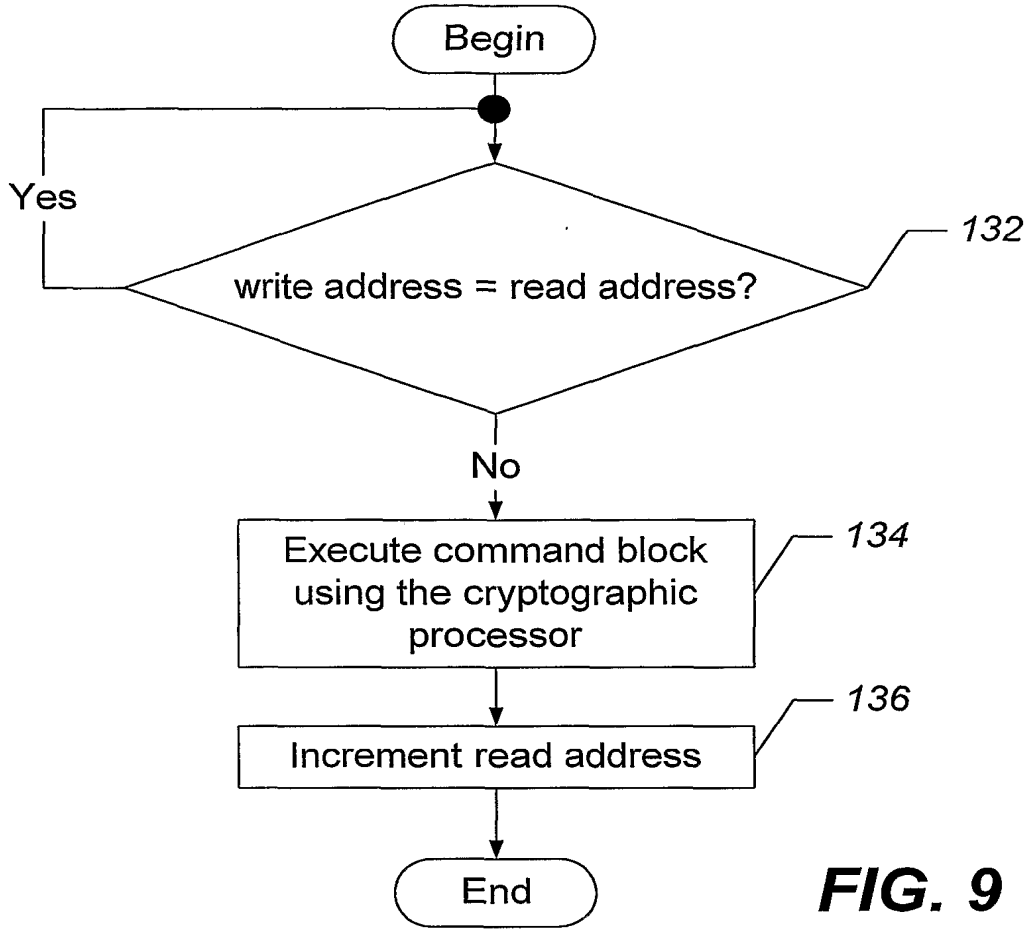
Begin
Yes
write address = read address?
132
No
Execute command block using the cryptographic processor
134
Increment read address
136
End

FIG. 9

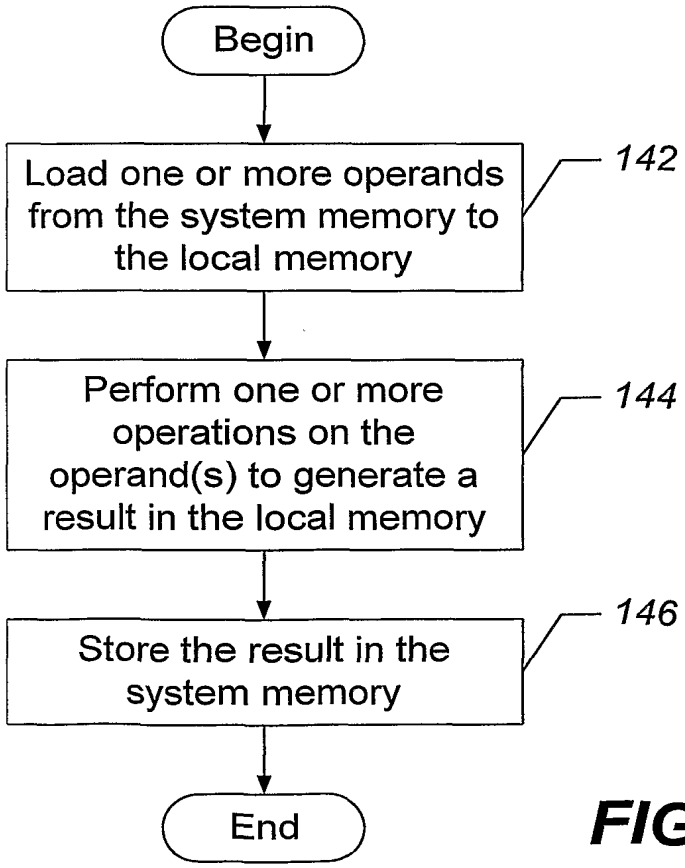
Begin
Load one or more operands from the system memory to the local memory
142
Perform one or more operations on the operand(s) to generate a result in the local memory
144
Store the result in the system memory
146
End

FIG. 10

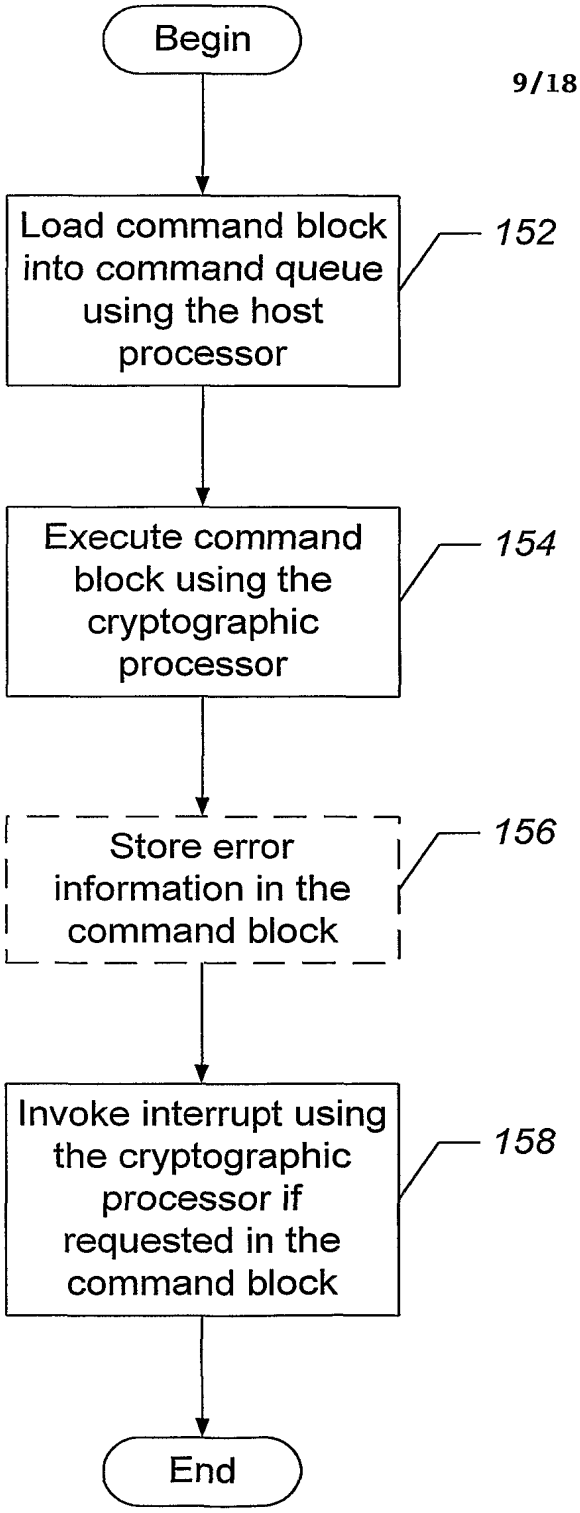
Begin
Load command block into command queue using the host processor
152
Execute command block using the cryptographic processor
154
Store error information in the command block
156
Invoke interrupt using the cryptographic processor if requested in the command block
158
End

FIG. 11

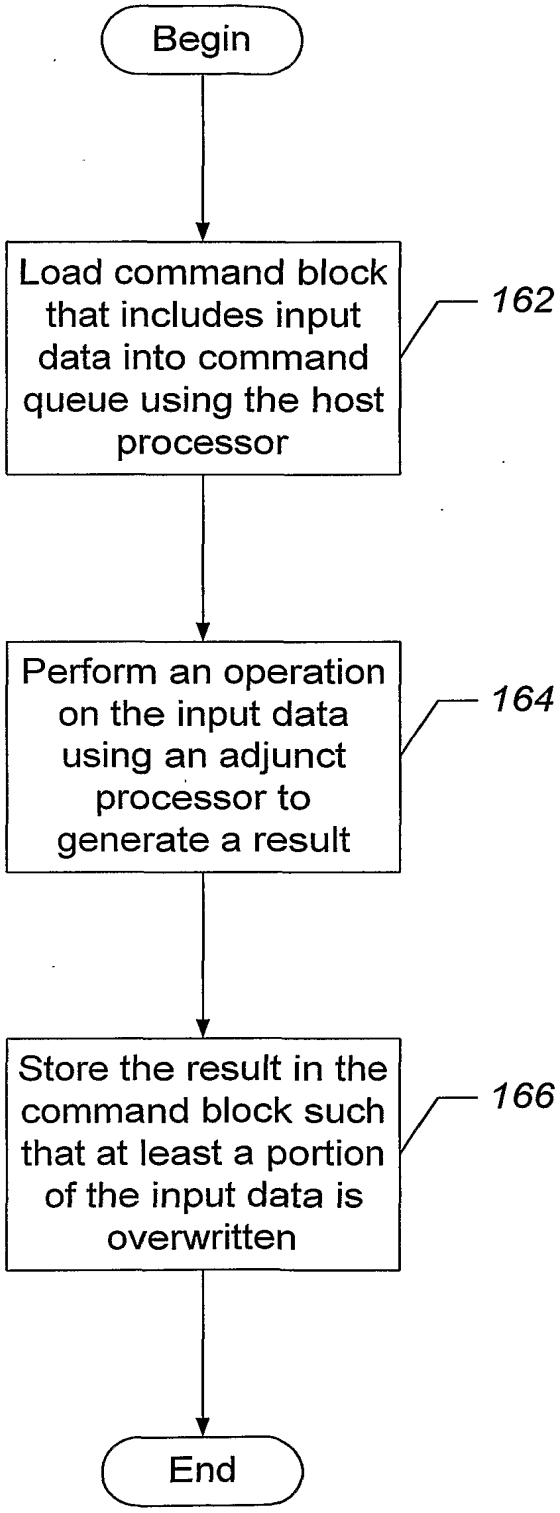
Begin
Load command block that includes input data into command queue using the host processor
162
Perform an operation on the input data using an adjunct processor to generate a result
164
Store the result in the command block such that at least a portion of the input data is overwritten
166
End

FIG. 13

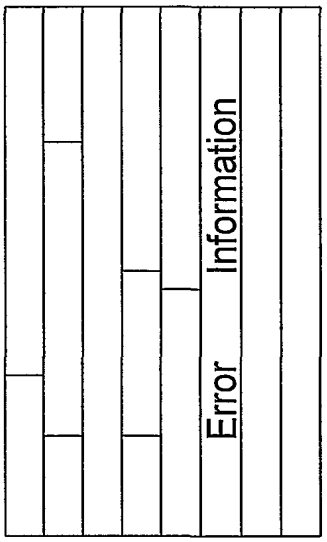

FIG. 12B

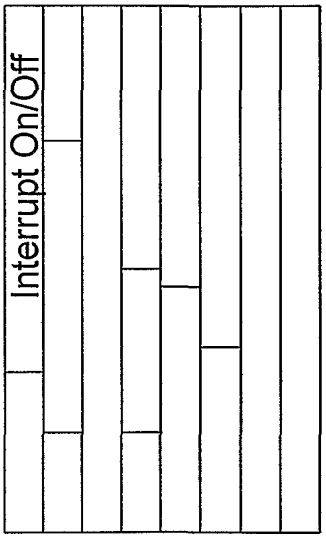

FIG. 12A

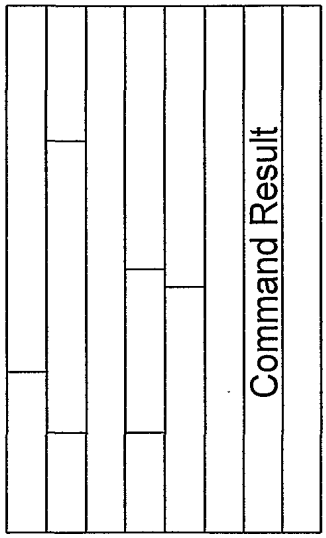

FIG. 12D

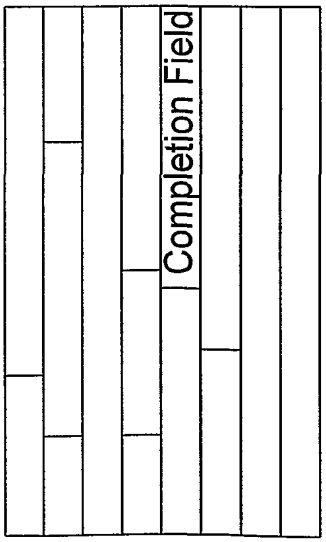

FIG. 12C

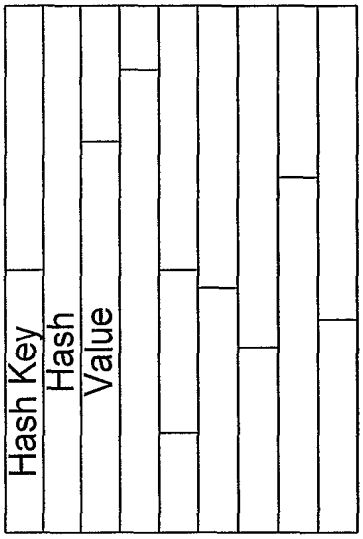

FIG. 14B

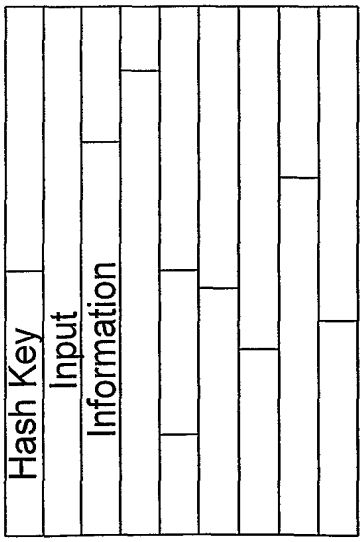

FIG. 14A

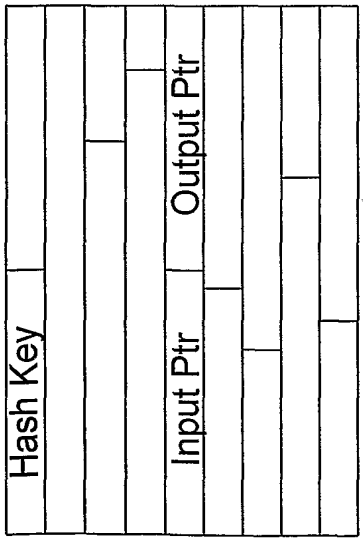

FIG. 15

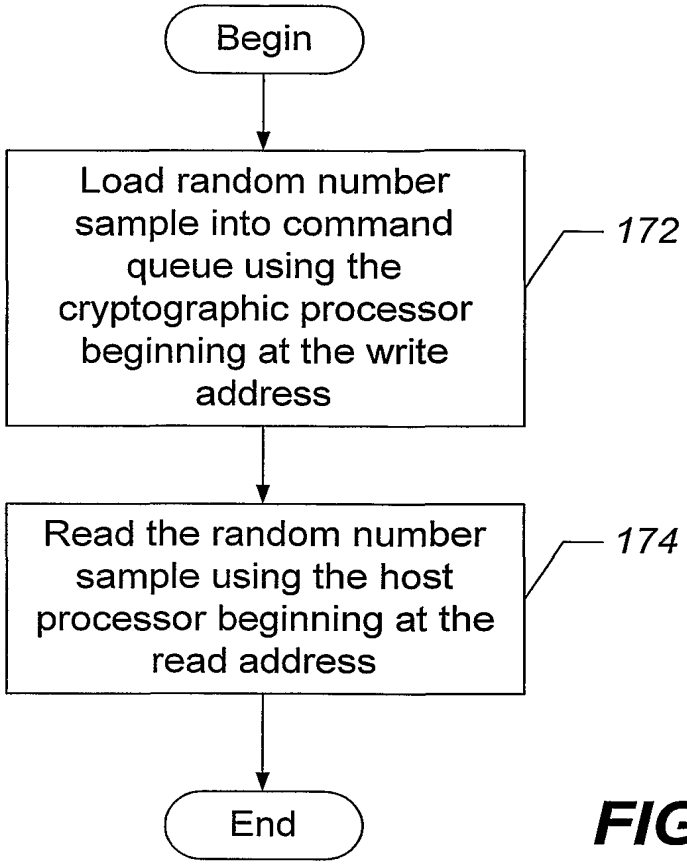
Begin
Load random number sample into command queue using the cryptographic processor beginning at the write address
172
Read the random number sample using the host processor beginning at the read address
174
End

FIG. 16

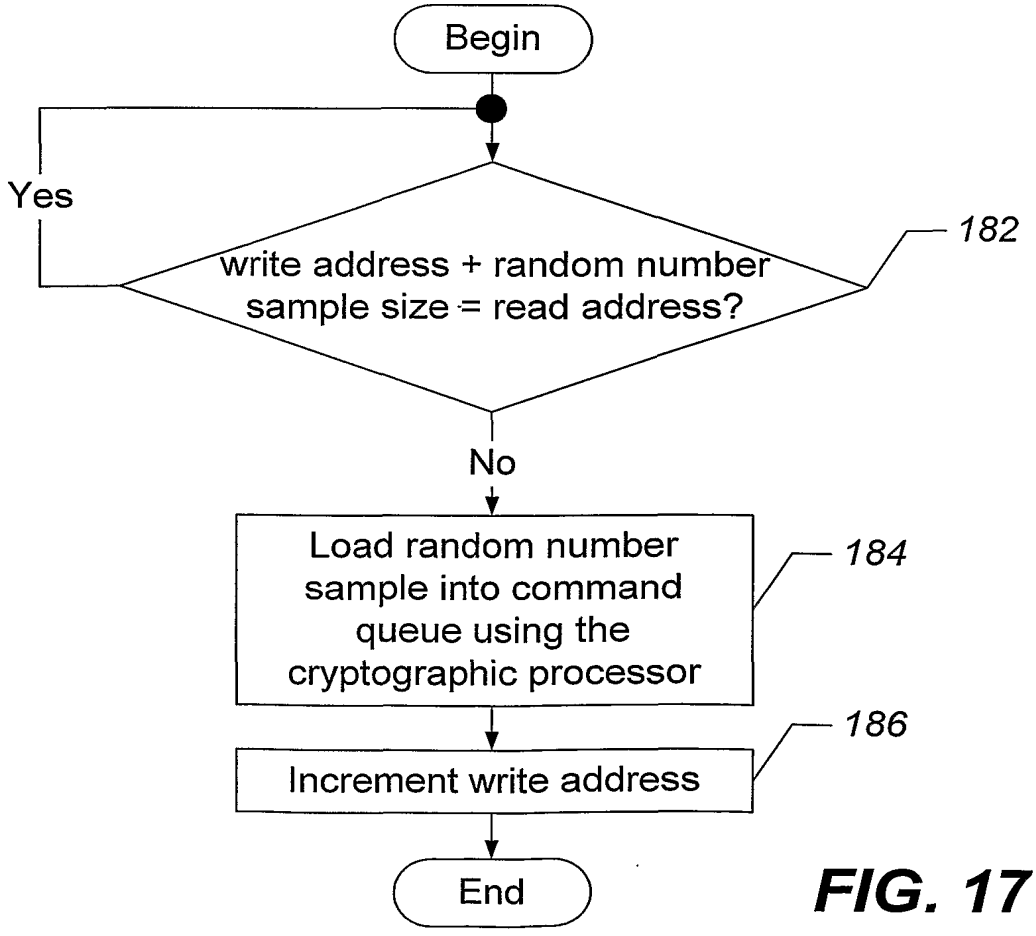
Begin
Yes
write address + random number sample size = read address?
182
No
Load random number sample into command queue using the cryptographic processor
184
Increment write address
186
End

FIG. 17

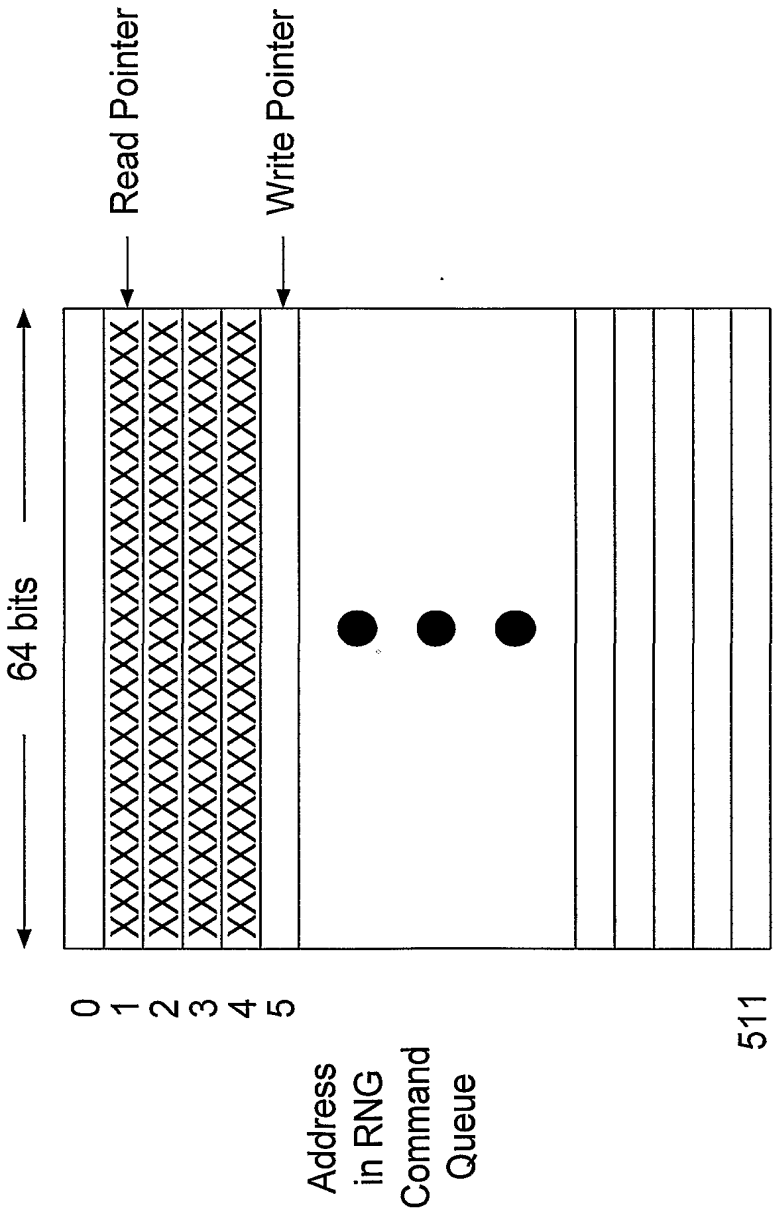
Read Pointer
Write Pointer
64 bits
0
1
2
3
4
5
511
Address in RNG Command Queue

FIG. 18

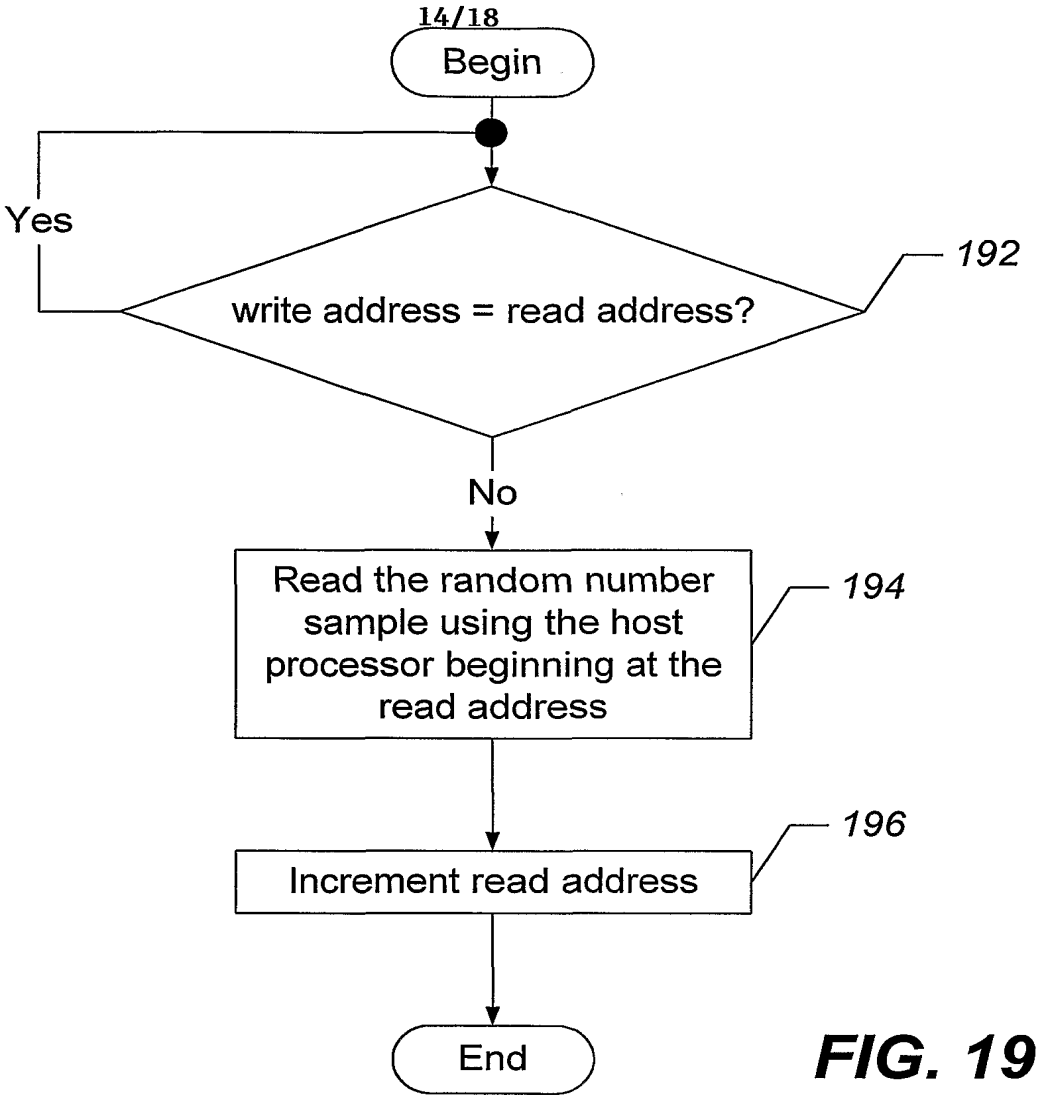
Begin
Yes
write address = read address?
192
No
Read the random number sample using the host processor beginning at the read address
194
Increment read address
196
End

FIG. 19

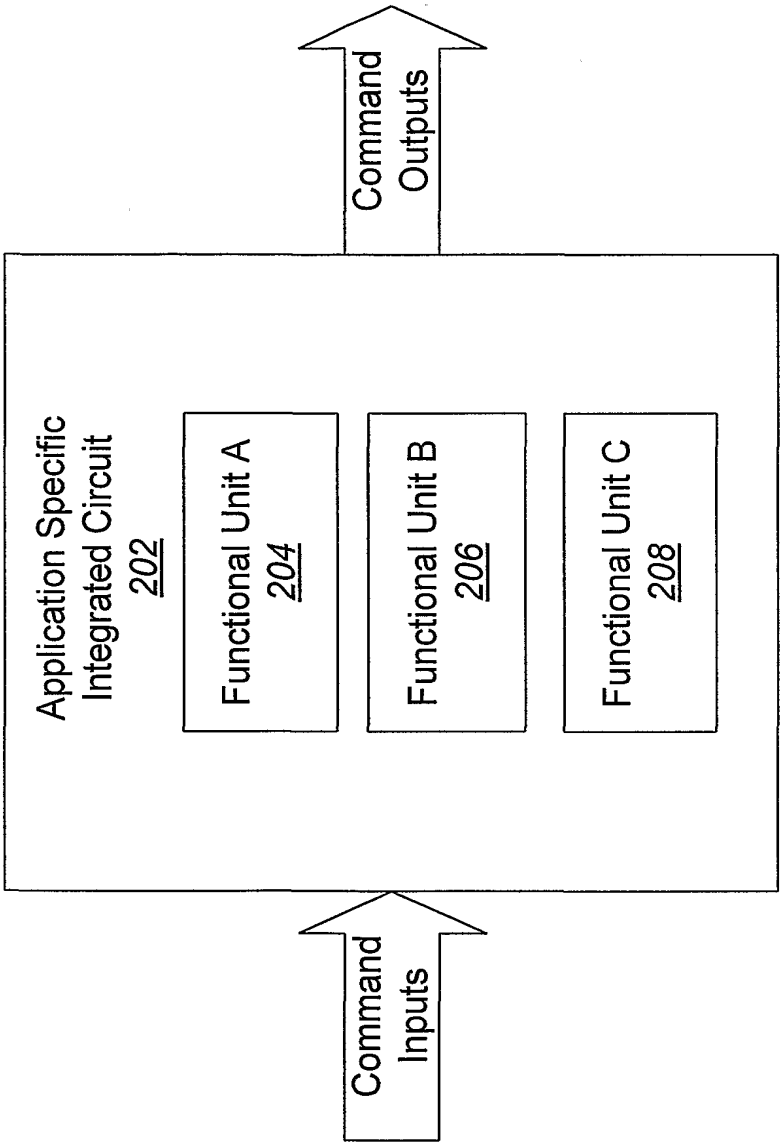

FIG. 20 (Prior Art)

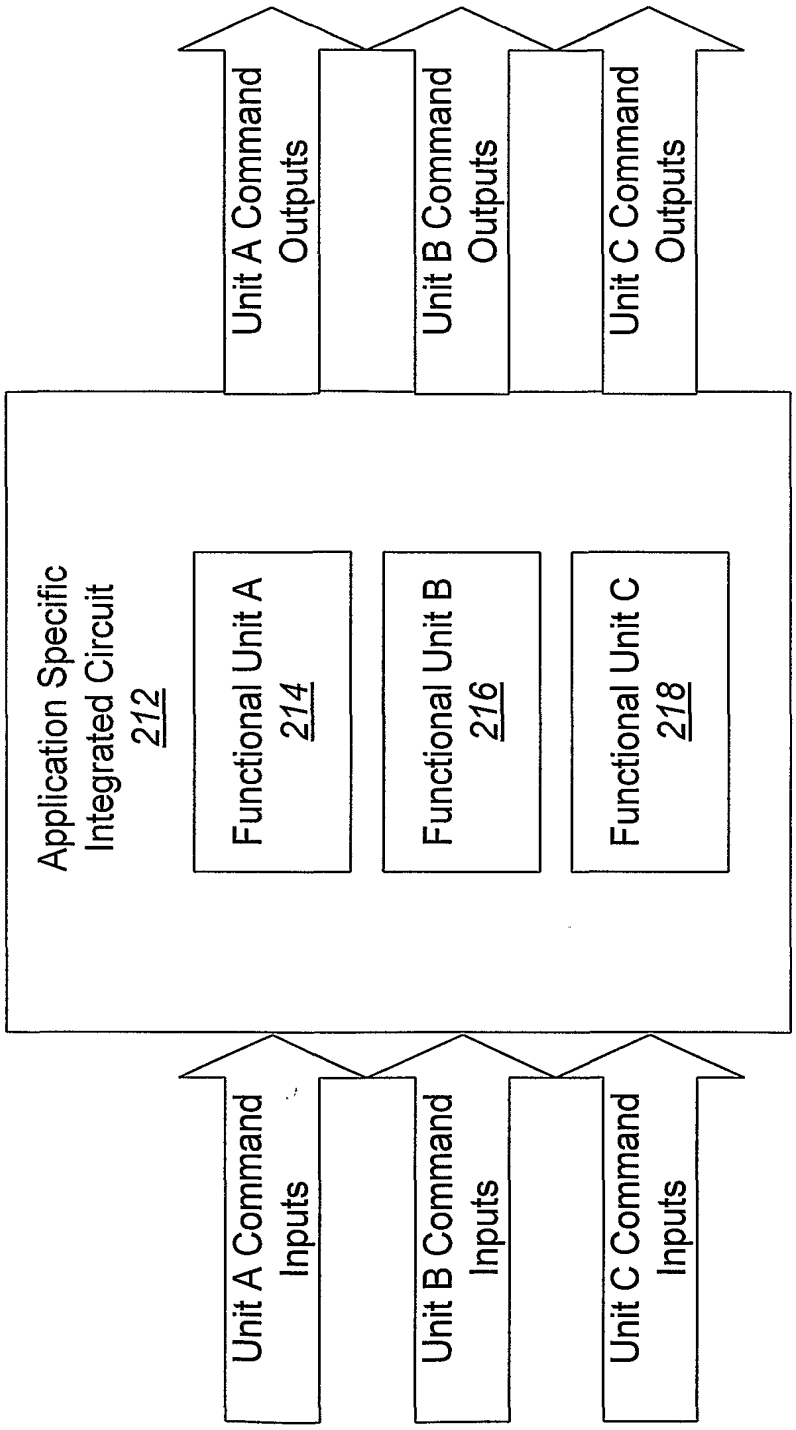
Unit A Command Inputs
Unit B Command Inputs
Unit C Command Inputs
Application Specific Integrated Circuit 212
Functional Unit A 214
Functional Unit B 216
Functional Unit C 218
Unit A Command Outputs
Unit B Command Outputs
Unit C Command Outputs

FIG. 21

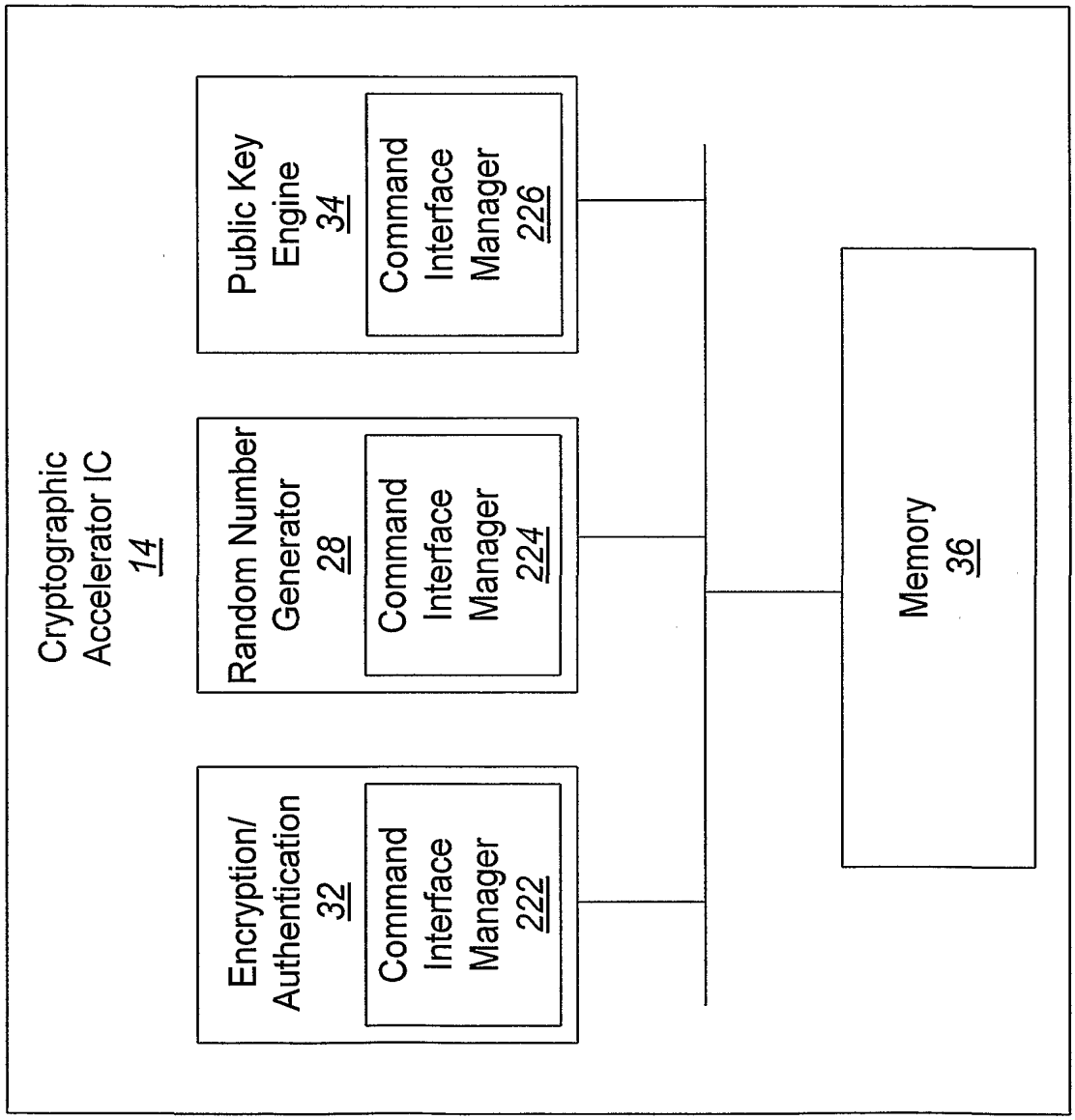
Cryptographic Accelerator IC 14
Encryption/ Authentication 32
Command Interface Manager 222
Random Number Generator 28
Command Interface Manager 224
Public Key Engine 34
Command Interface Manager 226
Memory 36

FIG. 22

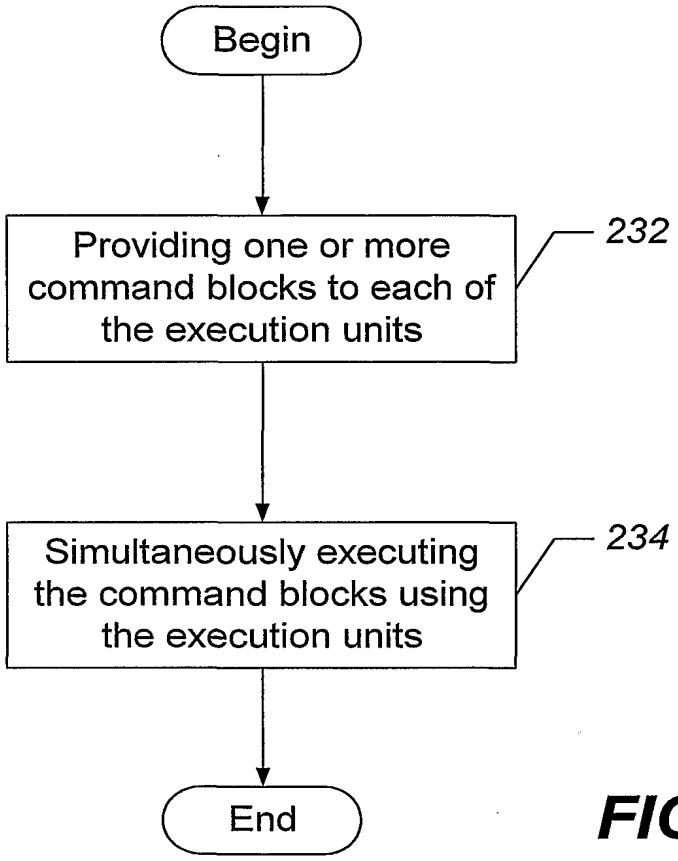
Begin
Providing one or more command blocks to each of the execution units
232
Simultaneously executing the command blocks using the execution units
234
End

FIG. 23